Philipps-Universität Marburg
Fachbereich Mathematik und Informatik

Dissertation mit dem Thema
„Modellierung und Animation von computergenerierten Pflanzen“

eingereicht von Markus Vincon im
Oktober 2008
Betreuer: Prof. Dr. M. Sommer

Inhaltsverzeichnis

Zusammenfassung

In dieser Arbeit geht es um das Modellieren und die Animation von computergenerierten Pflanzen. Besonderer Wert wird dabei auf die Kombination der beiden Punkte Modellieren und Animation gelegt. Das Ziel ist es, ein System zu beschreiben, welches sowohl ein intuitives Modellieren von computergenerierten Pflanzen als auch ein einfaches Modellieren der Animationsaspekte erlaubt. Dabei wird von Anfang an ein besonderer Wert auf eine sehr effiziente Umsetzung der Ideen und Verfahren gelegt, da die Effizienz der Schlüssel für ein auf diesem Gebiet erfolgreiches Verfahren ist. Insbesondere im Hinblick auf den Zusammenhang zwischen computergenerierten Pflanzen und virtuellen Welten wird gezeigt, wie diese effizient miteinander verknüpft werden können. Obwohl die Effizienz im Vordergrund steht, wird in dieser Arbeit auch viel Wert auf mögliche Verschönerungen des Ergebnisses und auf den Einbezug von möglichst vielen Parametern und Mechanismen, die das Aussehen und die Animation von computergenerierten Pflanzen betreffen, gelegt.

Die Entwicklung des Systemes (der sogenannte PlantAnimator) erfolgte dabei in verschiedenen Schritten, die jeweils auf den Erfahrungen und Ergebnissen der vorhergehenden Schritten aufbauen konnten. Dabei wurde zu Beginn auf zwei bestehende Ansätze aufgebaut bzw. diese miteinander kombiniert. Im Folgenden wurden die Ansätze und die darin enthaltenen Ideen aber wesentlich verbessert und erweitert.

Im Laufe der Arbeit stellte sich zudem heraus, dass das ganze Gebiet insbesondere durch die Querbeziehungen mit den virtuellen Welten äusserst umfangreich ist. Daher finden sich gleich eine ganze Reihe von interessanten Fragestellungen und Punkten in dem Ausblick am Ende dieser Arbeit.

Kapitel 1

Einführung

1.1 Motivation

Die Grundidee für diese Arbeit entstand aus dem Gedanken, ob für die Modellierung und die Animation von computergenerierten Pflanzen nicht ein Verfahren durch eine Kombination aus regelbasierter Objekterzeugung und differentiellen L-Systemen genutzt werden kann. Bei der regelbasierten Objekterzeugung wird eine Pflanze anhand von gegebenen Regeln aus verschiedenen Modulen (zum Beispiel für Stamm, Äste und Blätter) sehr benutzerfreundlich und intuitiv zusammengesetzt. Bei den differentiellen L-Systemen werden die Änderungen von gewissen Parametern durch in die Grammatik (die sonst die Regeln, wie sie auch in der regelbasierten Objekterzeugung verwendet werden, beschreibt) eingearbeitete Differentialfunktionen beschrieben.

Dabei sollte von Anfang an Wert auf eine möglichst effiziente Umsetzung des Konzeptes gelegt werden, da die Effizienz das entscheidende Kriterium zu dem Einsatz von Verfahren in Film- und Spiele-Industrie ist (oft wird diese viel stärker bewertet als beispielsweise die botanische oder physikalische Korrektheit des Verfahrens). Der daraus entstandene PlantAnimator setzt daher sehr stark auf affine Transformationen für die Umsetzung der Parameter während die Module auf Basis von Vertices definiert werden. Diese Konzentration auf affine Transformationen sollte von Anfang an zwei Optimierungs-Konzepte für grosse virtuelle Welten oder die Darstellung von grossen Pflanzen-Populationen und Ökosystemen berücksichtigen. Die Geometrie-Informationen der Module (also die Vertices) werden für den Einsatz innerhalb einer Pflanze nur durch Transformationen wie Skalierungen, Rotationen oder Translationen verändert.

Die beiden Optimierungs-Konzepte sind:

- Durch ein Konzept der SharedGeometry in Verbindung mit sogenannten Links sollte es möglich sein, viel Speicherplatz zu sparen. Damit sollte auch das Laden und Anlegen von neuen Modulen im Hauptspeicher (zum Beispiel durch Pflanzenwachstum) sehr effizient zu sein.
- Das Konzept der Quantisierung sollte dadurch ebenfalls sehr effizient umsetzbar sein. Von der Quantiesierung bei computergenerierten Pflanzen oder der Definition von virtuellen Welten spricht man, wenn man mit wenigen einzeln modellierten Pflanzen eine grosse Szene füllt. Dazu werden die einzelnen Pflanzen mehrfach in der Szene plaziert, wobei sie sowohl gedreht als auch skaliert werden können (also das Anwenden von weiteren Transformationen auf die Pflanze).

Danach beschäftigt sich die Arbeit mit der Frage, ob und wie man das Konzept der umwelt-getriebenen Animation (das Musterbeispiel dafür ist die Bewegung, die ein Wind erzeugt, der durch einen Baum weht) in den bestehenden PlantAnimator integrieren kann. Das zu erreichen, ohne sich die angedachten Optimierungsmöglichkeiten zu zerstören, ist ein wesentliches Ziel dieser Arbeit. Im Laufe der Arbeit zeigte sich weiterhin, dass ein effizientes Level of Detail Konzept für die Animation von den generierten Pflanzen möglich sein sollte. In Verbindung mit einer effizienten Berechnung des Blickfeldes, einer möglichen guten Movement Prediction in allgemeinen virtuellen Welten (also kein Verfahren, was mit Trainingsdaten für eine spezielle Szene angelernt wird) und eines für das Modellieren eines Ökosystemes ausgereiften Raumaufteilungsverfahrens, sollte es dann möglich sein, das Level of Detail Konzept für die Animation umzusetzen. Eine Erweiterung oder der letzte Schritt des Level of Detail für die Animation ist dann die Lazy Animation. Diese bedeutet, dass eine Pflanze, die gar nicht sichtbar ist, auch keinen Rechenaufwand für das Aktualisieren zugeteilt bekommt. Lazy Animation kann nur dann gut funktionieren, wenn das Aktualisieren der vorher inaktiven Pflanzen möglichst effizient ablaufen kann, sobald diese wieder in das Blickfeld kommen, aber auch dies bietet der Ansatz des PlantAnimators.

1.2 Aufbau der Dissertation

In den folgenden Kapiteln werden zuerst die verwendeten Grundlagen auf dem Gebiet des Modellierens von computergenerierten Pflanzen und des Modellierens von Pflanzenwachstum beschrieben (Kapitel 2 bis 4). In Kapitel 5 wird eine erste Version des Kernkonzeptes dieser Arbeit beschrieben, der PlantAnimator.

Es wird versucht zu zeigen, wie einfach das gemeinsame Modellieren sowohl der Pflanze als auch des Pflanzenwachstumes sein kann. Kapitel 6 beschreibt dann ein Konzept, welches für die Darstellung von computergenerierten Pflanzen in virtuellen Welten sehr wichtig ist, und welches bisher noch recht wenig in der Wissenschaft bearbeitet wurde. Danach wird geprüft, wie sich das Konzept der umwelt-getriebenen Animation so in den bestehenden PlantAnimator integrieren lässt, dass mögliche angedachte und für die Definition von virtuellen Welten sehr wichtige Optimierungsmöglichkeiten nicht negativ beeinflusst werden. In Kapitel 7 geht es dann darum, einige Dinge aus der erste Version des PlantAnimators zu verbessern, wie zum Beispiel die Kontrolle der Zeitachse und das Modellieren von Modul-Übergängen und der Geometrie dieser Übergänge, oder sogar ganz neue Konzepte wie Rekursion bei der Modellierung oder die umwelt-getriebene Animation einzuarbeiten. In Kapitel 8 geht es zuerst um die effiziente Umsetzung von Blickfeld-Berechnungen und ein intelligentes Raumfaufteilungsverfahren zum Modellieren eines ganzen Ökosystemes (mit der Erweiterung um das Berechnen von Einflüssen zwischen den Pflanzen, wie zum Beispiel durch das Konkurieren um Licht). Danach werden ebenfalls in Kapitel 8 noch einmal die Möglichkeiten eines Level of Detail Konzeptes für die Animation und die Lazy Animation beschrieben. Zum Schluss wird in Kapitel 9 eine Zusammenfassung der Arbeit sowie ein Ausblick auf weiter mögliche Forschungspunkte geboten.

Kapitel 2

Grundlagen

In den Grundlagen-Kapiteln wollen wir uns zuerst mit den botanischen Grundlagen beschäftigen. In den folgenden Kapiteln wird es dann um die Möglichkeiten zur Modellierung von Pflanzen und um die Modellierung insbesondere in Bezug auf die Animation des Pflanzenwachstumes unter Berücksichtigung von verschiedensten Umwelt-Faktoren gehen. Viele Aspekte der Botanik wie beispielsweise zum Beispiel der Zusammenhang zwischen Struktur und Funktion einer Pflanze werden dabei nicht berücksichtigt. Hauptsächlich werden sogenannte kormophytische Pflanzen (Bäume und Sträucher) betrachtet, die bestimmend für viele Landschaften sind.

2.1 Botanische Grundlagen

Kormophytische Pflanzen bestehen aus Wurzeln, Blättern und der Sprossachse. Die Sprossachse bestimmt durch ihre Ausrichtung im Raum und durch die Form der Verzweigungsbildung massgeblich das Aussehen der Pflanzen. Daneben bestimmen noch die verschiedenen Prozesse der Knospung, der zugrundeliegenden Raumaufteilung und Tropismen, siehe auch 2.1.2 und 4.8, das Aussehen einer Pflanze.

2.1.1 Aufbau von Pflanzen

Die Wurzel oder der Keimling ist die Grundlage für die Pflanze, wird aber im Kontext der computergenerierten Pflanzen oft nicht betrachtet, da diese nicht sichtbar unter der Erdoberfläche sind. Wenn man aber doch die Wurzeln modellieren wollte, dann würden sich ähnliche Algorithmen wie für die nach oben

wachsenden Äste anbieten. Dazu müsste man das ganze nur in einer anderen Richtung ablaufen lassen und es auch etwas mehr in Abhängigkeit von weiteren Umweltfaktoren wie zum Beispiel der Bodenbeschaffenheit setzen.

Eine Pflanze oder deren Spross entwickelt sich ausgehend vom Keimling durch Längenwachstum. Ein wichtiger Aspekt des Sprosses ist die Gabelung (siehe auch 2.1.1.2). Während der Spross wächst, entstehen neue Sprossanlagen (Knospen), die wiederum Grundlage für neue Sprosse und die dadurch entstehende Verzweigungsform und Raumaufteilung sind.

2.1.1.1 Knospung

Wie schon oben geschrieben, entstehen Knospen als Grundlage für weitere Verzweigungen während des Längenwachstums von einem Spross. Bei den Knospen wird zwischen Gipfelknospen und Seitenknopsen (auch Achselknospen genannt) unterschieden.

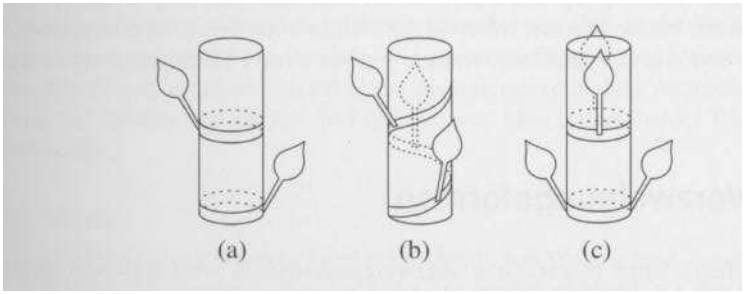


Abbildung 2.1: Blattstellungen (aus [Deu03])

Der Spross wird dabei in Knoten (Nodus) und Internodien-Bereich eingeteilt, Blätter entstehen dann aus dem ringförmigen Nodus-Bereich. Im Bild 2.1 kann man ein paar typische Formen von Blattstellungen sehen. Der Winkel zwischen benachbarten Blättern bleibt dabei aber nach der so genannten Äquidistanz-Formel immer konstant. In obigem Bild sehen wir (a) Distichie, (b) Dispersion und (c) Dekussation.

Distichie: Bei der Distichie wird pro Knoten genau ein Blatt erzeugt. Diese stehen aufeinanderfolgend jeweils um 180 Grad versetzt.

Dispersion: Hier ist der Divergenzwinkel kleiner und es ergibt sich eine schraubige Blattstellung. Die Winkel liegen meist im Bereich von 135 bis 144 Grad. Der Vorteil liegt in dem Minimieren der gegenseitigen Abschattung durch aufeinanderfolgende Blätter.

Dekussation: Hier werden an einem Knoten mehrere Blätter gebildet. Die Blätter aufeinander folgender Knoten stehen auf Lücke (auch Alternanz genannt), so dass ein günstiges Abschattungsverhältnis aufeinander folgender Blätter erreicht wird. Werden zwei Blätter pro Knoten erzeugt, dann sind diese jeweils um 90 Grad zu den Blättern des davorliegenden Knotens versetzt.

2.1.1.2 Verzweigungsformen

Für das Gesamtbild der Pflanze spielt neben den Blattstellungen insbesondere die Art der Verzweigung, die sich dann bei dem Hauptspross beginnend rekursiv in alle Seitenzweige fortsetzt, eine wichtige Rolle.

Bei der Verzweigungsform wird zu allererst nur unterschieden, ob der Hauptspross oder die Seitenzweige dominant sind. Wenn das Wachstum des Hauptsprosses dominant ist, dann spricht man von monopodialer Form (siehe auch Abbildung 2.2 (a)). In diesem Fall ist der Hauptspross länger als die Seitenzweige erster Ordnung, welche wieder länger sind als die Seitenzweige zweiter Ordnung. Dies ergibt eine kegelförmige Wuchsform. Sind die Seitenzweige dominant, so ergibt sich die sympodiale Wuchsform, welche für viele Laubbäume und Sträucher bezeichnend ist. Stehen bei der sympodialen Wuchsform die Seitenzweige entsprechend einer Dekussation und sind sie gleichwertig, so spricht man von Dichasium (siehe auch Abbildung 2.2 (b)), ist hingegen ein Seitenzweig dominant, so spricht man von einem Monochasium (siehe auch Abbildung 2.2 (c)).

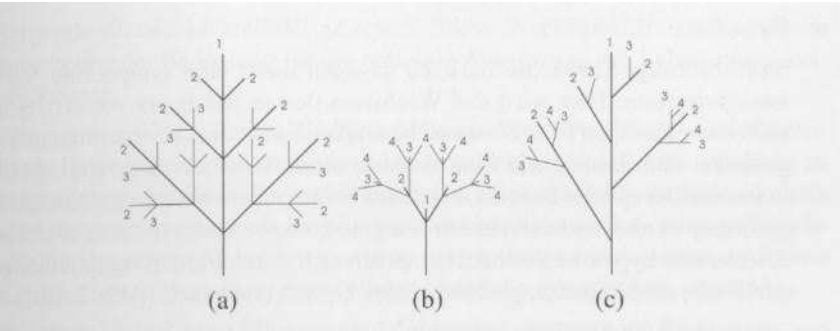


Abbildung 2.2: Verzweigungsformen (aus [Deu03])

2.1.2 Attraktoren und Inhibitoren

Es gibt verschiedene weitere Faktoren, die das Wachstums und Verhalten von Pflanzen wesentlich beeinflussen und die beim Modellieren von Pflanzen und ihrem Wachstum auch berücksichtigt werden sollten.

Zuallererst versucht jede Pflanze, den ihr zur Verfügung stehenden Raum optimal zu nutzen. Der Haupt-Trieb wächst zu diesem Zwecke in die Höhe und die Sekundärtriebe eher in die Horizontale. Dieses Verhalten hat grosse Ähnlichkeiten mit dem weiter unten noch beschriebenem Gravitropismus. Grundsätzlich wird mit dem Wort Tropismus in der Botanik die reizbedingte Orientierung eines Pflanzenorgans zum Reiz hin (positiver Tropismus, der einen Attraktor beschreibt) oder von diesem weg (negativer Tropismus, der einen Inhibitor beschreibt) bezeichnet.

Aus [And06]: Pflanzen behalten ihre Wachstumsfähigkeit ein Leben lang, was ihr gutes Anpassungsvermögen an äussere Umstände erklärt. Die hier betrachteten kormophytische Pflanzen verfügen über Gewebe, das aus teilungsfähigen Zellen besteht. Dieses so genannte Bildungsgewebe befindet sich an den Sprossspitzen sowie in der Schicht zwischen Rinde und Holz und ist für das Längen- und Dickenwachstum von Pflanzen verantwortlich. Einjährige Pflanzen wachsen hauptsächlich in die Höhe und sehr wenig im Durchmesser, wobei mehrjährige Pflanzen zudem auch noch im Durchmesser wachsen.

Damit hat die Pflanze also die Möglichkeit, auf die verschiedenen Tropismen, die durch Außenreize hervorgerufen werden und Verkrümmungen bzw. Deformationen der Sprossachsen verursachen, zu reagieren. Das Aussehen einer Pflanze wird also neben der Verzweigungsart auch von den Verformungen der Sprossachsen bei den Wachstumsbewegungen bestimmt.

Wendet sich das Organ zum Reiz hin, handelt es sich um einen positiven Tropismus wendet es sich ab, handelt es sich um einen negativen Tropismus. Orientiert sich das Organ in einem bestimmten Winkel zur Reizrichtung, wird der Tropismus als plagiotrop bezeichnet. Eine tropistische Krümmung erfolgt dabei durch verschieden starkes Streckenwachstum auf der reizzu- und reizabgewandten Seite. Bei einem positiven Tropismus zum Beispiel wächst die reizabgewandte Organfläche stärker wie man in Abbildung 2.3 sehen kann.

Es existieren eine Reihe von Tropismen, die sich durch die Art des steuernden Reizes unterscheiden. Die größte Auswirkung auf die Erzeugung der Form einer Pflanze haben allerdings Photo- und Gravitropismus. Beim Phototropismus legt das Licht die Richtung der Bewegung fest. Die meisten Sprossachsen und viele Blattstiele sind positiv phototrop, wohingegen die Seitenzweige häufig plagiotrop sind. Die Blattspreiten stehen meist in einem rechten Winkel zum einfallenden

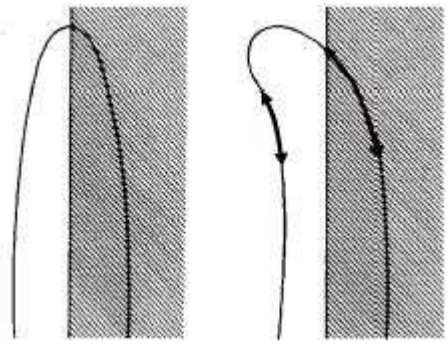


Abbildung 2.3: Unterschiedliches Wachstum der Seiten durch einen Tropismus zum Beispiel bei Licht aus [And06]

Licht. Ein weiteres Phänomen ist die Schattenflucht, bei der teilweise beleuchtete Blätter zur besonnten Seite hin verschoben werden. Daneben gibt es auch die umgekehrte Variante, bei der eine Pflanze vor zu großer Sonneneinstrahlung flieht.

Der steuernde Reiz bei dem Gravitropismus ist dagegen die Schwerkraft bzw. die Gravitation der Erde. Mit Hilfe von gravitropischen Wachstumsbewegungen kann sich eine Pflanze im Raum orientieren. So wachsen die Fichtenstämme an einem Steilhang genau in der Gegenrichtung der Erdanziehung, unabhängig von der Steigung des Hangs. Die Spross-Hauptachsen einer Pflanze zeigen einen negativen Gravitropismus, d.h. orientieren sich entgegen der Richtung der Erdanziehung. Seitenzweige zeigen Transversalgravitropismus, bei dem die Organe einen unterschiedlichen Winkel zum Schwerevektor einnehmen. Die gravitropischen Bewegungen der Pflanzenorgane sind, wie auch die phototropischen Bewegungen, auf verschieden starkes Wachstum zweier Organhälften zurückzuführen. Photo- und Gravitropismus können in vielen Fällen durch lokale Mechanismen approximiert werden. Bei einer negativ gravitropen Pflanze kann zum Beispiel die Ausrichtung eines neu generierten Elementes jeweils um ein Stück in die dem Schwerkraftvektor entgegen gesetzte Richtung verdreht werden.

Neben dem Stichwort Tropismus findet man in der Literatur auch den Begriff der Nastie, die beide oft nur schwer zu unterscheiden sind (siehe auch unter [Wik08c]).

Von Nastien wird gesprochen, wenn die Bewegungsrichtung durch den Bau des sich bewegenden Organs bestimmt ist. Die Richtung aus welcher der Reiz erfolgt ist also nicht entscheidend. Der Reiz dient nur als Signal für eine festgelegte Bewegung. Nastien vollziehen sich relativ schnell und können reversibel sein.

Vertreter dieser Art sind zum Beispiel Thermonastie (eine Blüte öffnet oder

schliesst sich anhand der Temperatur) oder Photonastie (eine Blüte öffnet oder schliesst sich anhand der Sonneneinstrahlung).

Als dritten Begriff neben Nastie und Tropismus finden sich noch die Taxien. Taxien sind freie Ortsbewegungen, die durch einen Außenfaktor bestimmt sind. Im Pflanzenreich sind diese aber bis auf wenige Ausnahmen bei begeißelten oder amöboiden Einzellern, bei Kolonien und bei einzelligen Entwicklungsstadien höher organisierter Formen (Gonosporen, Zoosporen, Gameten) nicht zu finden.

Mit dem Begriff Hydrotaxis bezeichnet man zum Beispiel die Reaktion auf Feuchtigkeitsdifferenzen, die eigentlich für Pflanzen und insbesondere die Wurzelbildung auch interessant sein könnten.

2.2 Von der Umwelt beeinflusstes Wachstum

Mit dem Begriff des von der Umwelt beeinflussten Wachstumes möchte ich hier äussere Mechanismen zusammenfassen, die Einfluss auf das Wachstum und die Form einer Pflanze nehmen, bei denen aber noch nicht geklärt ist, inwieweit die Pflanze wirklich selbst darauf reagiert. Ein gutes Beispiel hierfür ist der Wind. Starker Wind hemmt das Wachstum und formt Bäume. Besonders windempfindlich ist zum Beispiel die Lärche, und deshalb sind Lärchen, die in Küstennähe stehen, oft säbelförmig. Man kann dies beispielsweise auch in Patagonien gut sehen. Alle Bäume dort sind durch den Wind in eine Richtung defomiert.

Im Gegensatz zu dem Begriff der umwelt-getriebenen Animation (siehe Kapitel 6) geht es hier aber nicht um den Wind, der eine Pflanze nur bewegt, aber darüberhinaus keinen Einfluss auf das Wachstum nimmt. Der hier betrachtete Wind könnte als eine Art besonderer Attraktor bzw. Inhibitor umgesetzt werden. Allerdings müsste man sich dabei noch Gedanken machen, wie man das Hemmen des Wachstumes modellieren kann. Dies ist sicher nicht ohne weiteres mit einem einfachen Attraktor oder Inhibitor erledigt.

Ein weiterer Aspekt dieses von der Umwelt beeinflussten Wachstumes ist das Beschädigen von Pflanzen durch Katastrophen wie Sturm. Während eines Sturmes können Äste abbrechen, was natürlich zu dem charakteristischen Aussehen eines Baumes beitragen kann. Wir werden im folgenden sehen, dass die Folgen solcher Katastrophen schon verschiedentlich in der bisherigen Literatur zur Modellierung von Pflanzen und Pflanzenwachstum vorgesehen sind.

Kapitel 3

Modellierung von Pflanzen

Pflanzen können auf unterschiedliche Arten und Weisen modelliert werden. In diesem Kapitel wollen wir einige der verschiedenen und ersten Verfahren betrachten. Beginnen wollen wir mit den zellulären Automaten, die schon Mitte der fünfziger Jahre des zwanzigsten Jahrhunderts durch John von Neumann erfunden wurden. Zelluläre Automaten hatten aber erst ab etwa 1966, als einer breiteren Schicht von Forschern Rechner zur Verfügung standen, ihre Blütezeit.

3.1 Prozedurale Methoden

Prozedurale Methoden sind parametriesierbare Algorithmen, die zur Erzeugung einer Art von Pflanzen, oft auch nur einer einzigen Spezies, dienen. Eines der ersten Beispiele für einen solchen Algorithmus sind die zellulären Automaten. Neben den zellulären Automaten fallen aber beispielsweise auch noch Darstellungen über Partikelsysteme oder über Fraktale unter den Oberbegriff „Prozedurale Methoden“.

3.1.1 Zelluläre Automaten

Für zelluläre Automaten wird der benutzte Raum (normalerweise ein gewöhnlicher zwei- oder dreidimensionaler Raum) in gleichförmige Zellen unterteilt. Die Zellen können zwar auch als Dreiecke, Tetraeder oder andere Grundobjekte modelliert werden, aber üblicherweise verwendet man Würfel. Nach der Aufteilung des Raumes in Zellen hat jede Zelle eine Anzahl fest definierter Nachbarn. Aus-

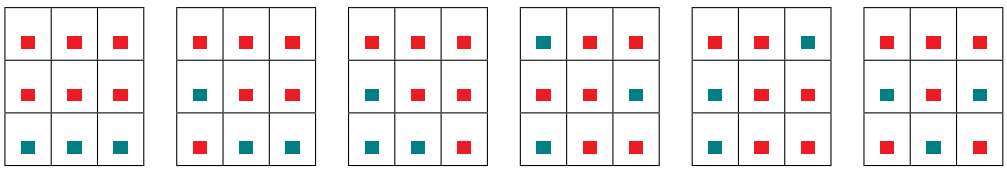
serdem wird jeder Zelle ein Anfangszustand zugewiesen, welcher sich im Laufe der iterativ voranschreitenden Zeit in Abhängigkeit der Zustände der Nachbarzellen verändern kann.

Ein recht bekanntes Beispiel für zelluläre Automaten ist das sogenannte Game of Life, welches von John Horton Conway 1970 entwickelt wurde (siehe auch [Wik08b]). Das zweidimensionale Spielfeld wird hierzu in Zeilen und Spalten unterteilt und jede sich daraus ergebende Zelle kann zwei Zustände annehmen (tot oder lebendig). Von Generation zu Generation kann sich nun der Zustand der Zellen ändern, dabei wird die Folgegeneration für alle Zellen gleichzeitig berechnet und ersetzt die aktuelle Generation. Der Zustand einer Zelle, lebendig oder tot, in der Folgegeneration hängt nur vom Zustand der acht Nachbarzellen dieser Zelle in der aktuellen Generation ab.

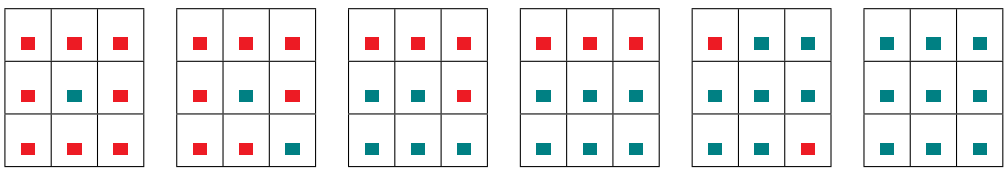
Die zu Beginn benutzen Regeln waren:

- Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren.
- Eine lebende Zelle mit genau drei lebenden Nachbarn bleibt in der Folgegeneration lebend.
- Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an Einsamkeit.
- Bei zwei lebenden Nachbarzellen behält die Zelle in der Folgegenerationen ihren Status.
- Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Überbevölkerung.

In den folgenden Beispielen wird die mittlere Zelle in der nächsten Generation geboren werden. Hierbei steht rot für tot und grün für lebend.



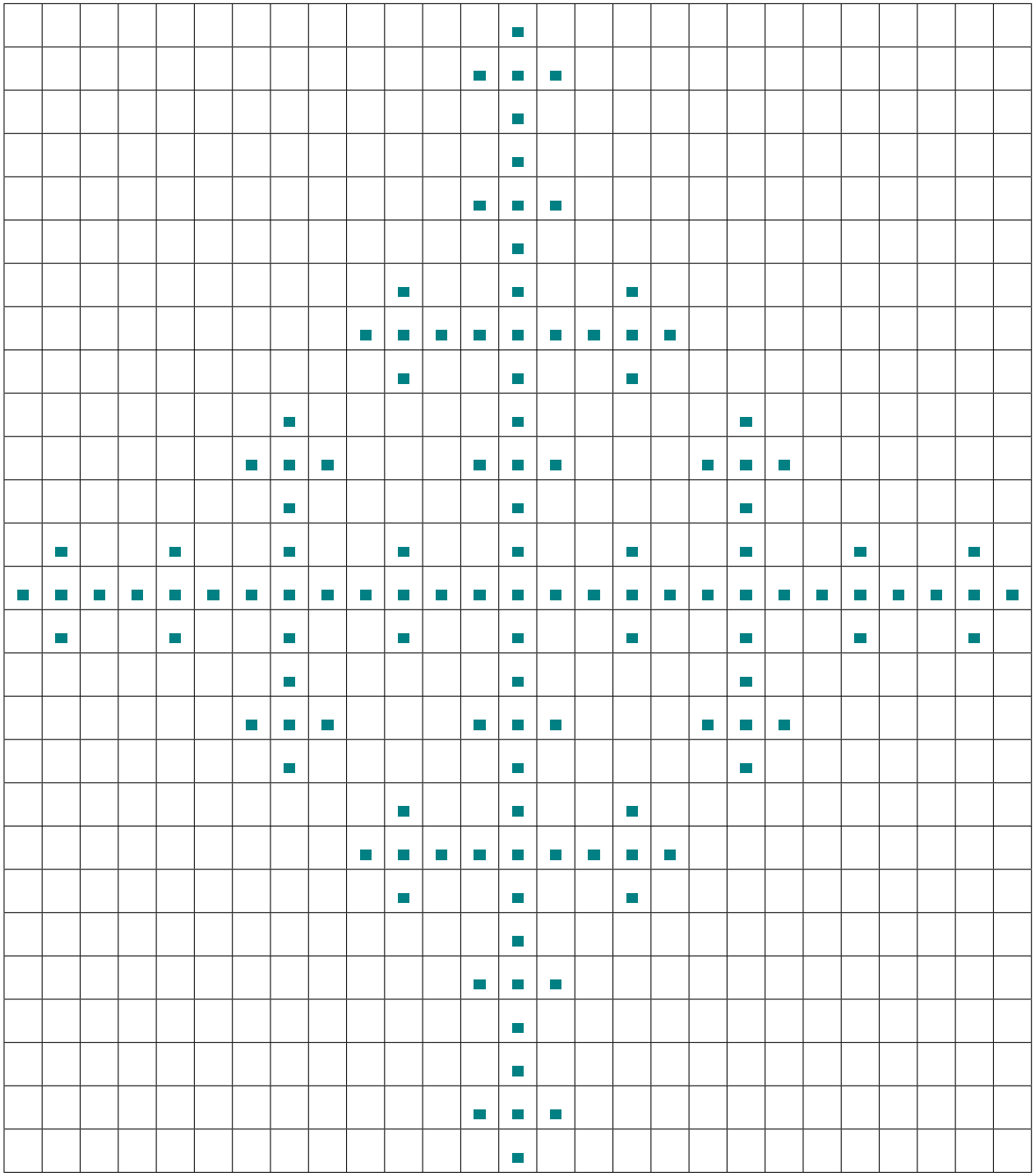
In den folgenden Beispielen würde die mittlere Zelle sterben.



Um jetzt mit Regeln eine für Pflanzen grundlegende Verzweigungsstruktur zu erzeugen, können wir uns zum Beispiel die folgenden beiden Regeln definieren:

1. Wir beginnen mit einer Anfangszelle in der Mitte
2. Nun werden nach und nach alle Zellen dazugenommen, die Nachbarn von schon dazugenommenen Zellen sind und
    - nicht von 2 oder mehr dazugenommenen Zellen berührt werden.
    - nicht von einer im gleichen Schritt dazugenommenen Zelle berührt werden.

Mit diesen Regeln lässt sich schon eine Verzweigungsstruktur wie folgt (nach 13 Schritten) erstellen:



Insgesamt lassen sich so schon einige grundlegende Verzweigungsstrukturen definieren und wenn dann noch ein paar Blätter dazukommen, hätte man auch eine Pflanze. Ein Nachteil an diesem wirklich einfachen Verfahren ist allerdings die schlechte Steuerbarkeit, das heisst, dass es bei dem Definieren von solchen Regeln nicht klar sein muss, wie sich die Pflanze dann wirklich entwickelt. Weitere

Nachteile sind die hohe Anzahl an Zellen, die man sicherlich für ein realistisches Aussehen benötigt, und die teilweise mangelhafte Animierbarkeit des Ganzen bedingt durch fehlende Geometrie- oder Zusammenhangs-Informationen zwischen den Zellen. Es wäre dann sicherlich höchst ineffizient, wenn man dieses auf der Basis der einzelnen Zellen animieren möchte. Wobei anzumerken ist, dass eine ganz einfache Wachstumsanimation (ohne äussere Einflüsse) anhand der unterschiedlichen Generationen sicherlich möglich ist.

Im Folgenden werden wir ein paar weitere Verfahren aus dem Bereich der prozeduralen Modellierung kennenlernen, wo in den Modellier-Prozess schon botanische Grundlagen mit eingegangen sind. Diese erlauben das intuitive Modellieren, welches als ein grosses Plus der prozeduralen Methoden angeführt wird, viel besser als die zellulären Automaten.

3.1.2 Kontinuierliche Modelle

Schon ab etwa 1967 entstanden erste kontinuierliche Modelle zum Modellieren und Darstellen von Bäumen oder mindestens Verzweigungsstrukturen. Der Vorreiter dieser Modelle war Cohen, der eine Verzweigungsstruktur als Folge diskreter Abschnitte aufbaut, die nach ein paar einfachen Regeln ausgerichtet werden.

Das Wachstum findet nur an den Spitzen statt. Der biologische Mechanismus des Sprosses wird dadurch direkt nachgebildet. Die Stärke und der Winkel des Wuchses werden durch die aktuelle Richtung, ein lokales Dichtefeld, dessen Gradienten sowie die Resistenz der Struktur gegen Winkelveränderungen bestimmt. Die Verzweigungstendenz wird durch ein probabilistisches Maß bestimmt, das neben einem generellen Wert von der Entfernung zur letzten Verzweigung und vom lokalen Dichtefeld abhängt.

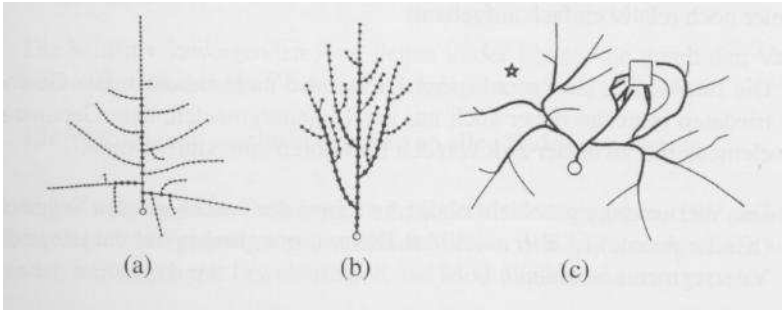


Abbildung 3.1: Einfache kontinuierliche Beispiele (aus [Deu03])

In Bild 3.1 finden wir ein paar Beispiele zu Verzweigungsstrukturen, die aus diesem Ansatz resultieren können. In (a) wird ein Beispiel mit Verzweigungsrestrik-

tionen (Punkt 3 der obigen Liste) gezeigt, (b) zeigt ein Beispiel zur räumlichen Ausrichtung (Punkt 2 der obigen Liste) und in (c) wird der Einsatz von Attraktoren und Inhibitoren gezeigt (in diesem Falle ist der Stern ein Inhibitor, während das Quadrat ein Attraktor ist).

Der Ansatz enthält auch die Änderung von Parameterwerten über die Verzweigungsordnung hinweg. Dies erlaubt überhaupt erst eine visuelle Differenzierung in Verzweigungstrukturen und wird in vielen nachfolgenden Ansätzen eingesetzt.

Aufbauend auf diesem Ansatz entwickelten dann Honda und Fisher vom botanischen Hintergrund kommend, die Verzweigungsstruktur von Bäumen und anderen Pflanzen. Hierbei entstehen einfach aufgebaute dreidimensionale Bäume nach folgendem Modell:

- Die Internodien sind gerade, ihre Dicke wird nicht beachtet. Die Geometrie-Daten bestehen daher aus Linien-Segmenten.
- Die Verzweigung geschieht binär, die Länge der verzweigenden Segmente (Kindsegmente) ist über zwei Verhältniswerte r_1 und r_2 auf die Länge des Vatersegmentes bezogen.
- Vatersegment und beide Kindsegmente liegen in einer Ebene, die Kindsegmente haben einen konstanten Verzweigungswinkel. Eine Ausnahme bilden die Abzweigungen von dem Hauptstamm, bei welchen Divergenz zugelassen wird.
- Die Verzweigung nimmt in diskreten Schritten mit jeder Verzweigungsordnung zu.

Ein Beispiel dazu befindet sich in Bild 3.2. Dort ist der Baum in (a) auf die XZ-Ebene projeziert, in (b) auf die YZ-Ebene und in (c) auf die XY-Ebene.

Dieses Verfahren wird dann wiederum von Aono und Kunii aufgegriffen und mit Augenmerk auf die Verzweigungen wie folgt erweitert:

- Verzweigung entsteht durch Bifurkation, die damit modellierten Bäume haben monopodiale oder sympodiale Form.
- Länge und Durchmesser der verzweigten Äste nehmen mit konstantem Faktor ab, die Verzweigungswinkel bleiben konstant über alle Verzweigungsstufen.
- Die beiden verzweigten Äste liegen in der Ebene, die durch seinen Vater und seinen maximalen Gradienten aufgespannt wird.

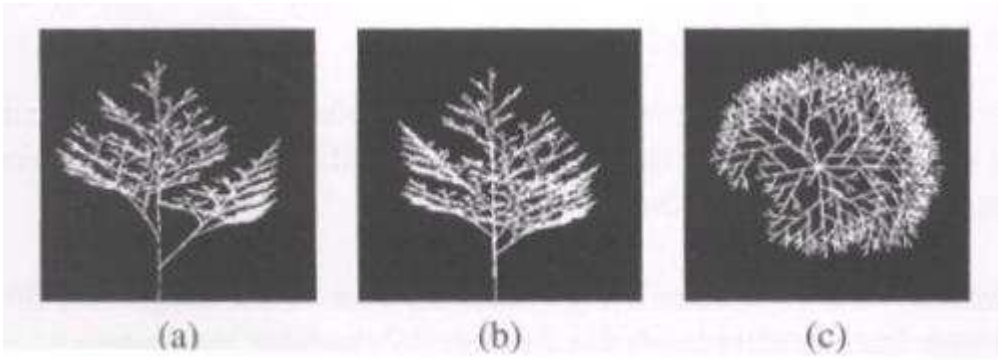


Abbildung 3.2: Einfache Bäume nach Honda und Fisher (aus citeDEU03)

- Die Verzweigung geschieht simultan an allen Enden der Zweige.
- Attraktoren und Inhibitoren werden eingeführt, um zum Beispiel Wind nachbilden zu können (siehe Abbildung 3.3 (c))
- Verzweigungswinkel können in Abhängigkeit zu ihrem Entstehungswinkel geändert werden (siehe Abbildung 3.3 (d))
- Weitere Verzweigungsarten werden eingeführt.

Ein Beispiel hierzu findet sich in Bild 3.3. Die Teilbilder (a) und (b) sind durch verschiedene Verzweigungswinkel entstanden, in (c) wird ein Inhibitor verwendet und in (d) wird ein variierender Verzweigungswinkel benutzt.

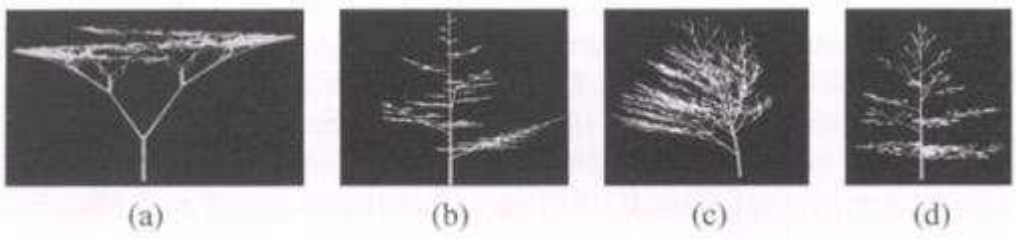


Abbildung 3.3: Verzweigungsstrukturen nach Aono und Kunii (aus [Deu03])

3.1.3 Partikel-Systeme

Anfang bis Mitte der Achtziger-Jahre befassten sich Reeves und Blau mit der Möglichkeit, Pflanzen und Pflanzenbewuchs unter Zuhilfenahme von Partikel-Systeme zu modellieren und darzustellen. Wobei die grundsätzliche Motivation nicht auf botanischer Korrektheit lag, sondern auf einem realistisch aussehendem Gesamtbild, welches dann auch in dem Film 'The Adventures of Andre and Wally B.' eingesetzt werden sollte. Ebenfalls mit Partikel-Systemen wurde bei der Feuerwand der Genesis-Bombe in dem Film Star Trek II - The Wrath of Khan gearbeitet (siehe auch [Ree83]), dort wurde eine zweistufige Hierarchie von Partikelsystemen benutzt. Ein erstes Partikel-System dessen Zentrum bei dem Einschlagpunkt der Bombe lag, hat Partikel erzeugt, die selbst wieder Partikel-Systemen waren und in konzentrischen Kreisen um das Zentrum herumlagen. Diese Partikel-Systeme der zweiten Stufe haben dann zeitlich aufeinander abgestimmt Partikel ausgestossen, die jeweils eine einzelne Explosion darstellte, so dass das Gesamtbild einer Feuerwand, die über den gesamten Planeten lief, entsprach.

In [Ree83] wird zuerst die Möglichkeit beschrieben, wie das Partikel-System auch gleich das grundsätzliche Aussehen einer Pflanze bestimmen kann. Dort wird ein Grashaufen durch ein Partikel-System modelliert, allerdings entsteht das Gras nicht durch das Zeichnen der Partikel, sondern indem man die Bewegungsbahnen der Partikel während ihrer Lebenszeit aufzeichnet und diese dann jeweils als einen Grashalm zeichnet.

In [RB85] wird dann die grundsätzliche Form der Pflanze eigentlich unabhängig von Partikel-Systemen vorgegeben und die Blätter werden mit Partikeln modelliert. Diese Vorgehensweise wird als strukturiertes Partikel-System bezeichnet. Die Pflanzen mit Partikel-Systemen bestehen also aus einem normal modellierten Stamm und Ästen (ohne Partikel-System) und an die Äste werden dann, wenn dort keine weiteren Äste abzweigen, verschieden gefärbte Partikel als Blätter angefügt. Das Modellieren von Stamm und Ästen wird hier schon recht aufwendig und mit einigen Parametern betrieben:

- Ein Grenzwert, bis zu welcher Rekursionstiefe das Erzeugen von Ästen fortgesetzt werden soll.
- Ein Grenzwert, bis zu welcher Dicke das Erzeugen von Ästen von fortgesetzt werden soll.
- Die Höhe eines Baumes, die sich aus einer Durchschnittshöhe und einem Zufallswert berechnet.

- Die Breites des Baumes und dabei auch die grundsätzliche Gestalt (zum Beispiel konisch oder eliptisch).
- Aus Höhe, Breite und Gestalt kann dann das sogenannte bounding volumen berechnet werden.
- Die Dicke eines Astes, die (nicht notwendigerweise linear) abnimmt je länger der Ast wird.
- Die maximale Länge eines Astes, die aus dem Abstand zum Rand des bounding volumen und einem zufälligen Faktor berechnet wird. Der Ast kann allerdings auch kürzer sein, je nachdem wie schnell er sich verjüngt.
- Die Anzahl der Verzweigungen, die bei einem dünnerem Ast höher ist, aber auch von der Art des Baumes abhängen.
- Die Dicke der Kinder in Bezug auf den Vater, also je grösser der Vater an der Abzweigungsstelle ist desto dicker ist auch der Sohn.

Abbildung 3.4: Strukturierte Partikelsysteme von Reeves (aus [RB85]

Die Bäume, die so entstehen, werden danach noch mit verschiedenen Algorithmen nachbearbeitet, die weitere natürliche Vorgänge nachbilden sollen, zum Beispiel

- Längerfristige Auswirkungen von Wind
- Längerfristige Auswirkungen der Richtung der Sonneneinstrahlung
- Schwerkraft

- Verschiedene Katastrophen, indem einige Äste entfernt werden

In dieser Arbeit wird ausserdem betrachtet, wie man das Wiegen von Grass in einer leichten Brise durch zufällige Variationen in den Partikel-Systemen bewerkstelligen könnte. Der Wind wird dazu als ein eigenes Partikel-System definiert, welches in vorgegebenen Zeitabständen jeweils eine Welle von Partikeln in der Szene starten lässt. Diese Partikel bewegen sich grob in die vorher eingestellte Richtung des Windes, können von dieser aber leicht abweichen. Durch dieses Vorgehen hat man zu jedem Zeitpunkt eine Moment-Aufnahme des Windes in der Szene und kann die sogenannte Wind Map erstellen. Diese gibt zu festgelegten Punkten, die als Netz über die Szene gelegt sind, auf der Karte an, wie stark der Wind dort ist. Die Stärke des Windes an einem dieser Punkte wird festgelegt durch die Anzahl der Wind-Partikel, die sich in der Nähe dieses Punktes befinden. Nachdem für jedes Frame der Animation eine dieser allerdings nur zweidimensionalen Wind Maps erstellt wurde, wird das Verhalten der Pflanzen (in dem Falle Gras) auf den Wind modelliert. Dabei wird wie folgt vorgegangen:

1. Die Achse der Beugung eines Grashalmes wird so bestimmt, dass diese senkrecht zu der Windrichtung ist und durch die Basis des Grashalmes geht.

2. Der Winkel der Beugung hängt von der Intensität des Windes ab und nimmt mit zunehmender Höhe des Grashalmes zu, d.h. der Grashalm ist am härtesten an seiner Basis.

3. Wenn gerade kein Wind herscht, dann bewegt sich der Grashalm langsam und sinusförmig zurück in seine Ausgangsposition.

4. Die Auswirkungen des Windes werden additiv behandelt. Dafür werden die berechneten Parameter für die nächsten Frames gespeichert und die Gesamtbeugung als Summe aller aktiven Beugungen berechnet.

Über den vierten Punkt lässt sich sicherlich streiten. Die leider nicht beantwortete Frage ist hier, wie die additive Behandlung von Wind genau aussieht. Hier sollte man mindestens eine Gewichtung bei der Addition vornehmen und nicht zu viele Frames in der Vergangenheit betrachten. Ausserdem wird nicht berücksichtigt, dass die resultierende Beugung nicht nur von der Höhe des Pflanzenlementes sondern ganz massgeblich von dem tatsächlichem Durchmesser und einigen Materialeigenschaften der Pflanze abhängen sollte.

Dies sind alles Fragestellungen sein, die wir in Kapitel 6 genauer betrachten werden.

3.1.4 Fraktale

Aus Wikipedia: Fraktal ist ein von Benoît Mandelbrot (1975) geprägter Begriff (lat. fractus: gebrochen, von frangere: brechen, in Stücke zerbrechen), der natürliche oder künstliche Gebilde oder geometrische Muster bezeichnet, die einen hohen Grad von Skaleninvarianz bzw. Selbstähnlichkeit aufweisen. Das ist beispielsweise der Fall, wenn ein Objekt aus mehreren verkleinerten Kopien seiner selbst besteht. Geometrische Objekte dieser Art unterscheiden sich in wesentlichen Aspekten von gewöhnlichen glatten Figuren.

Aufgrund der Selbstähnlichkeit der Fraktale bietet es sich an, diese über Rekursion zu beschreiben und in [Opp86] wird von Oppenheimer ein Baum auch wie folgt beschrieben:

```
 1 tree :==
 2 {
 3         Draw Branch Segment
 4         if(too small)
 5                 Draw Leaf
 6         else
 7         {
 8                 # Continue to Branch
 9                 {
10                         Transform Stem
11                         tree
12                 }
13                 repeat n times
14                 {
15                         Transform Branch
16                         tree
17                 }
18         }
19 }
```

Dies ist ein Fraktal, da durch die Rekursion für Stamm und Äste nur ein einzelnes (speziell transformiertes) Segment benutzt wird. Insgesamt präsentiert Oppenheimer in dem Papier ein Modell mit den folgenden Eigenschaften, auf die wir gleich noch näher eingehen wollen:

1. Detaillierte Parameterisierung der Beziehungen zwischen tree nodes
2. Real Time Design und Animation von Baum-Bildern

3. Anwendung von stochastischem Modellen für die topologischen und geometrischen Parameter

4. Stochastisches Modellieren der Baum-Rinde

5. High Resolution (2024 x 1980) Shaded 3D Renderings

Der erste Punkt beschreibt, dass hier zwischen Topologie und Geometrie getrennt wird. Die Geometrie bezeichnet die Form und das Aussehen der einzelnen Teil-Objekte (tree nodes), während die Topologie unter anderem die Lage dieser Teil-Objekte zueinander bezeichnet. Durch die detaillierte Parameterisierung der Beziehungen zwischen den einzelnen Objekten kann damit das Aussehen der gesamten Pflanze auf einfache Art beeinflusst werden.

Ein Baum wird als eine Folge von Segmenten aufgefasst zwischen denen mit Transformationen die Lage der Segmente zueinander bestimmt wird, eine Vorgehensweise, die auch die effiziente Animation wesentlich unterstützt. Dies ist auch die Grundlage für den zweiten Punkt der oberen Liste und gilt insbesondere, wenn man einige der heute gängigen Entwicklungsumgebungen für 3D-Programmierung betrachtet, die mit einem Szene-Graphen arbeiten, also genau diese Aufteilung zwischen Topologie und Geometrie gut umsetzen können.

Der vierte Punkt der oberen Liste beschreibt die Vorgehensweise die Unebenheiten der Rinde des Baumes mit sogenannten „fractal noise“ wie zum Beispiel der fraktalen brownschen Bewegung zu bestimmen und diese dann mit „bump mapping“ auf die einzelnen Teile von Stamm und Ästen zu übertragen. Der fünfte Punkt der oberen Liste wird nicht näher ausgeführt, hängt aber sicherlich mit der effizienten Umsetzung der Pflanzen zusammen, die die für komplexes Rendering benötigte Zeit senken sollte.

Der Vorteil des Anwendens von stochastischen Modellen für die Parameter nach dem dritten Punkt der oberen Liste liegt auch auf der Hand, weil so mit einer einzigen Beschreibung unterschiedlich aussehende Bäume berechnet werden können. Zu den benutzten Parametern gehören

- Der Winkel zwischen Stamm und Ästen
- Das Grössenverhältnis zwischen Stamm und Ästen
- Die Rate mit der der Stamm kleiner wird
- Der Anteil an spiralförmige Biegungen in den Ästen
- Die Anzahl der Äste an jedem Stamm-Segment

In Abbildung 3.5 sieht man ein Beispielsbild aus dem Artikel, allerdings ohne Stochastik und daher regelmässig.

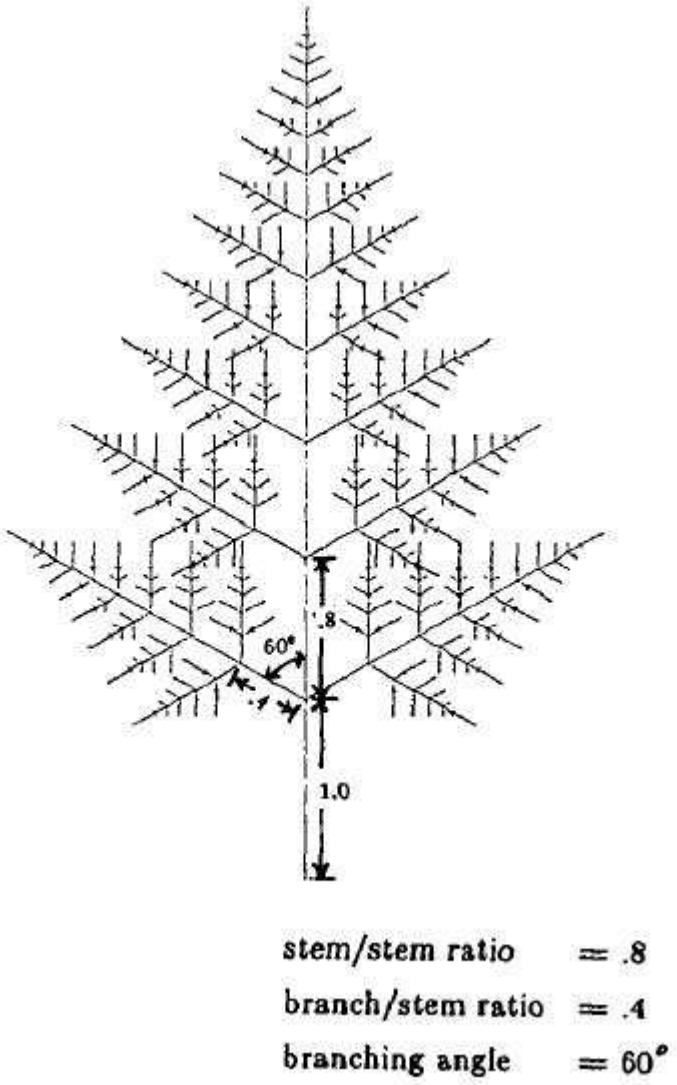


Abbildung 3.5: Beispiels-Baum von Oppenheimer (aus [Opp86])

In Abbildung 3.6 sieht man ein Beispielsbild mit Texturen.

Oppenheimers Vorschlag für eine Wachstums-Animation besteht darin, die Größe des Baumstammes zu interpolieren und den Abbruchswert für die Rekursion beim Erstellen des Baumes schrittweise zu erhöhen. Noch realistischer würde die Animation werden, wenn man weitere Parameter mit einbezieht, so schreibt er auch, dass viele Pflanzen sich entkräuseln, während sie wachsen.

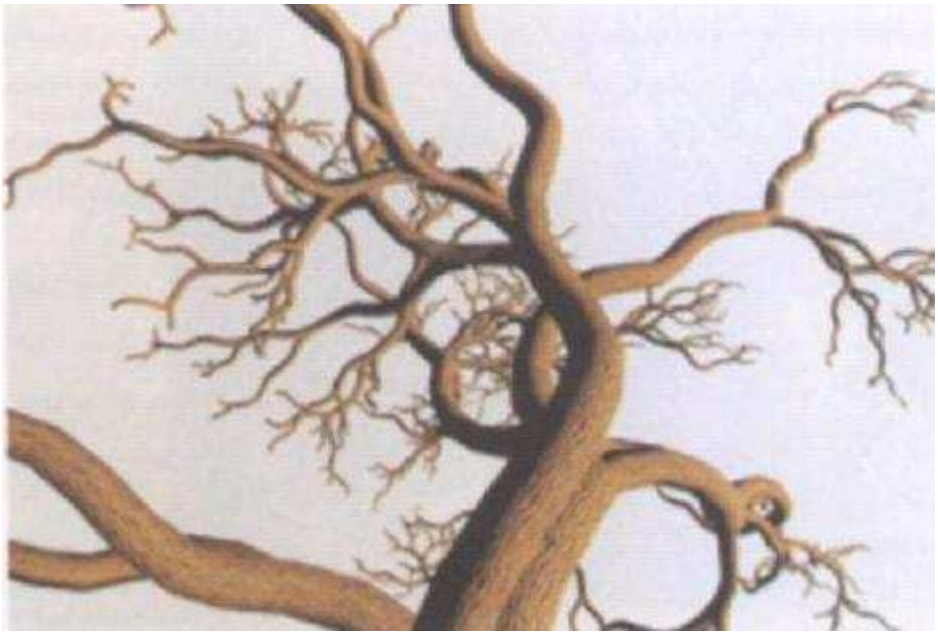


Abbildung 3.6: Ein weiterer fraktaler Baum von Oppenheimer (aus [Opp86])

In [CS06] wird dann ein Verfahren vorgeschlagen, welches vom Namen her die Überleitung zu dem nächsten Abschnitt sein könnte. Und auch, wenn die Verfasser durchaus auf Lindenmayer und Prusinkiewicz verweisen, so werden dort L-Systeme in keiner Weise so angewendet, wie sie in den nächsten beiden Abschnitten vorgestellt werden. Die Verfasser schlagen nur den Einsatz von L-Systemen oder IFS (iterierte Funktionen-Systeme) zur Erzeugung der fraktalen Geometrie vor. Bei iterierten Funktionen-Systemen werden mehrfach (meistens kontrahierende) Funktionen auf ein Grund-Objekt und Kopien dieses Grund-Objektes angewendet, um ein Gesamtbild zu erzeugen. Eigentlich wird in dieser Veröffentlichung nichts anderes als eine Weiterführung des Ansatzes von Honda inklusive der Weiterentwicklungen von Aono und Kunii, siehe auch Kapitel 3.1.2, vorgestellt. Also eine prozedurale Methode, wenn auch sicher gut parametriesiert und unter Berücksichtigung von vielen stochastischen Aspekten.

3.2 Regelbasierte Modellierung

In diesem Abschnitt wollen wir kurz grundsätzlich das regelbasierte Modellieren von Pflanzen vorstellen und insbesondere den Ansatz von Aristid Lindenmayer, Pflanzen über Textersetzungsysteme zu beschreiben.

In der Praxis könnten auch graphbasierte Ersetzungsverfahren verwendet werden, aber die Textersetzungssysteme haben sich bisher durchgesetzt und sind auch schon vielfach um neue Ideen erweitert worden. Einige dieser Ideen, insbesondere wenn es um das Wachstum von Pflanzen, deren Interaktion mit der Umwelt und die Animation des Wachstumes geht, werden wir im nächsten Kapitel genauer betrachten.

Die durch Lindenmayer definierten Textersetzungssysteme heissen L-Systeme und sind Quadrupel G = (V, S, w, P), wobei

- V die Zeichen enthält, die als Variable angesehen werden sollen.
- S die Zeichen enthält, die als Konstanten angesehen werden sollen. Die Zeichen aus V und S bilden das Alphabet des L-Systems.
- w ein Wort über dem Alphabet ist, das als Startwort oder Axiom des L-Systems bezeichnet wird.
- P eine Menge von geordneten Paaren aus Wörtern über dem Alphabet ist, welche Ersetzungsregeln definieren. Ist das erste Wort eines jeden Paares ein einzelner Buchstabe aus V, und zu jeder Variablen eine Ersetzungsregel bekannt, so spricht man von einem kontextfreien L-System, sonst von einem kontext-sensitiven.

Über solch ein L-System und eine grafische Interpretation der Buchstaben des Alphabetes lassen sich auch viele Fraktale beschreiben, das bekannteste Beispiel dafür ist die Koch'sche Kurve bzw. Schneeflocke. Hierbei gilt

- V={F}
- S={+,-}
- w=F–F–F
- P={(F → F+F–F+F)}

Zur grafischen Interpretation eines solchen L-Systems benötigt man einen Stauchungsfaktor $s < 1$ und einen Winkel a. Mittels des Stauchungsfaktors wird die Weglänge bei Rekursionstiefe n als s^n bestimmt Die grafische Interpretation kann dann wie folgt aussehen:

- F : Bewegung nach vorn um Länge s^n und Zeichnen eines Striches
- + : Drehung nach links, gegen Uhrzeigersinn, um den Winkel a
- - : Drehung nach rechts, im Uhrzeigersinn, um den Winkel a

Auf dem Bild 3.7 kann man sehen, wie sich die Schneeflocke im Laufe der verschiedenen Generationen (vorgenommenen Ersetzungen) entwickelt.

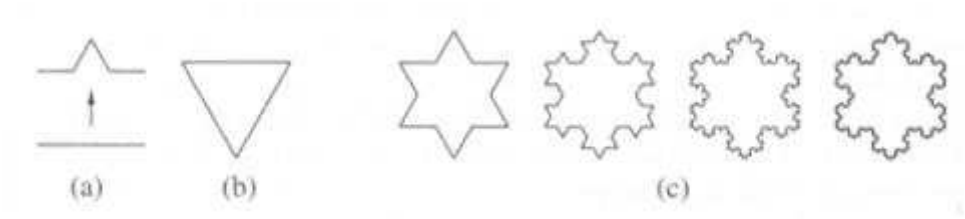


Abbildung 3.7: Die Schneeflocke von Koch (aus [Deu03])

In (a) ist die Ersetzungsregel grafisch angezeigt (jede Gerade wird durch eine Gerade mit Zacke ersetzt) und in (b) der Startzustand. In (c) werden dann die ersten 4 Generationen durchgespielt, man kann sehr schön sehen, wie sich die regelmässige Schneeflocke entwickelt.

Mit einem parametriesierbaren L-System (siehe Abbildung 3.8) können dann schon dreidimensionale Pflanzen erstellt werden (siehe Abbildung 3.9).

Diese Darstellung und Modellierung eignet sich allerdings noch nicht zur Animation. Zu diesem Zwecke könnte lediglich über die Anzahl der Generationen iteriert werden. Diese liegt bei dem gezeigten Beispiel aber nur bei 5 und ist daher ungeeignet, um die vielen für eine Animation notwendigen Zwischenbilder zu erzeugen.

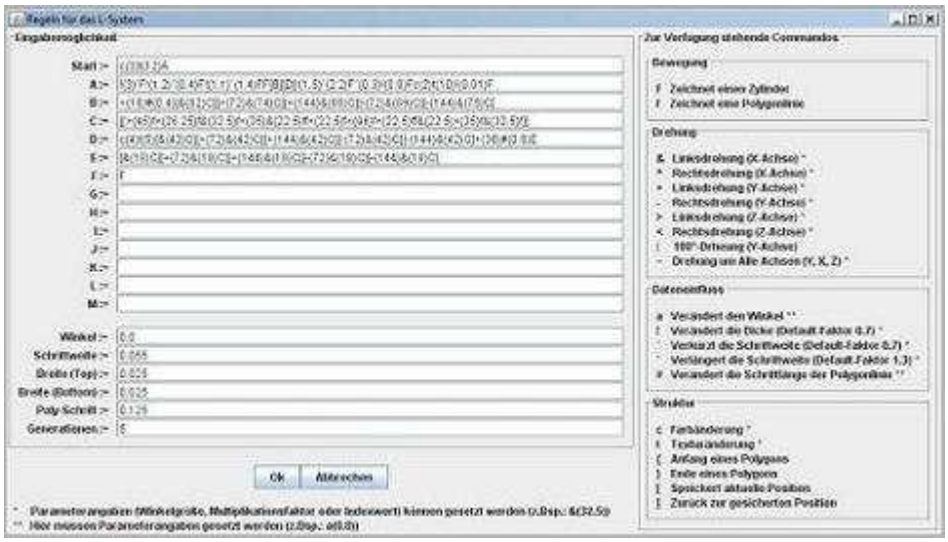


Abbildung 3.8: Ein Beispiel für ein parametriesierbares L-System

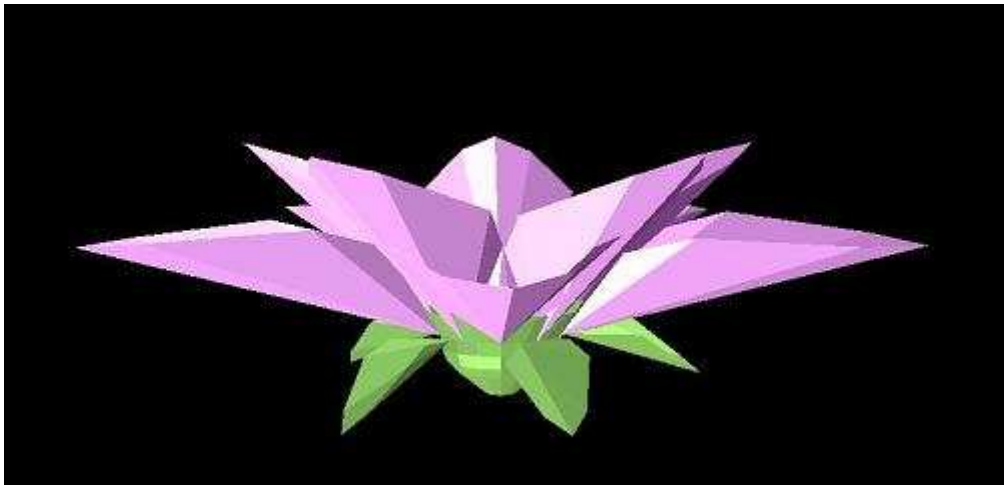


Abbildung 3.9: Das Bild, welches aus den Regeln in Abbildung 3.8 erzeugt wird

Dies soll an dieser Stelle als grundlegende Beschreibung von L-Systemen genügen, wir werden uns im nächsten Kapitel noch viel eingehender mit unterschiedlichen Ausprägungen und den Möglichkeiten von L-Systemen beschäftigen.

Als Überleitung zu dem nächsten Abschnitt wollen wir hier noch kurz das in dem Buch ‘Algorithmic Beauty of Plants‘ (siehe auch [PL96]) vorgestellte virtuelle Labor zur Modellierung von Pflanzen betrachten. Einen Beispielsschirm dieses virtuellen Labors sieht man in Abbildung 3.10. Das virtuelle Labor besteht hier aus einer Vielzahl von Programmen, die ineinander greifen und zusammen arbei-

ten. Auf dem Bild sehen wir rechts einige Bedienelemente, in der Mitte unten befindet sich ein Fenster von ISE (Interactive Surface Editor) in dem gerade ein Blütenblatt modelliert wird, dieses Blütenblatt wird dann in eine Sonnenblumen-Blüte eingesetzt (Mitte oben), die dann wiederum in eine ganze Pflanze eingesetzt wird (Links oben). Es bietet also Möglichkeiten zum grafischen Arbeiten und ist schon kein rein regelbasierter Ansatz mehr, obwohl die Verfasser versuchen, dass die Formate zum Austausch von Informationen überwiegend L-Systeme sind.

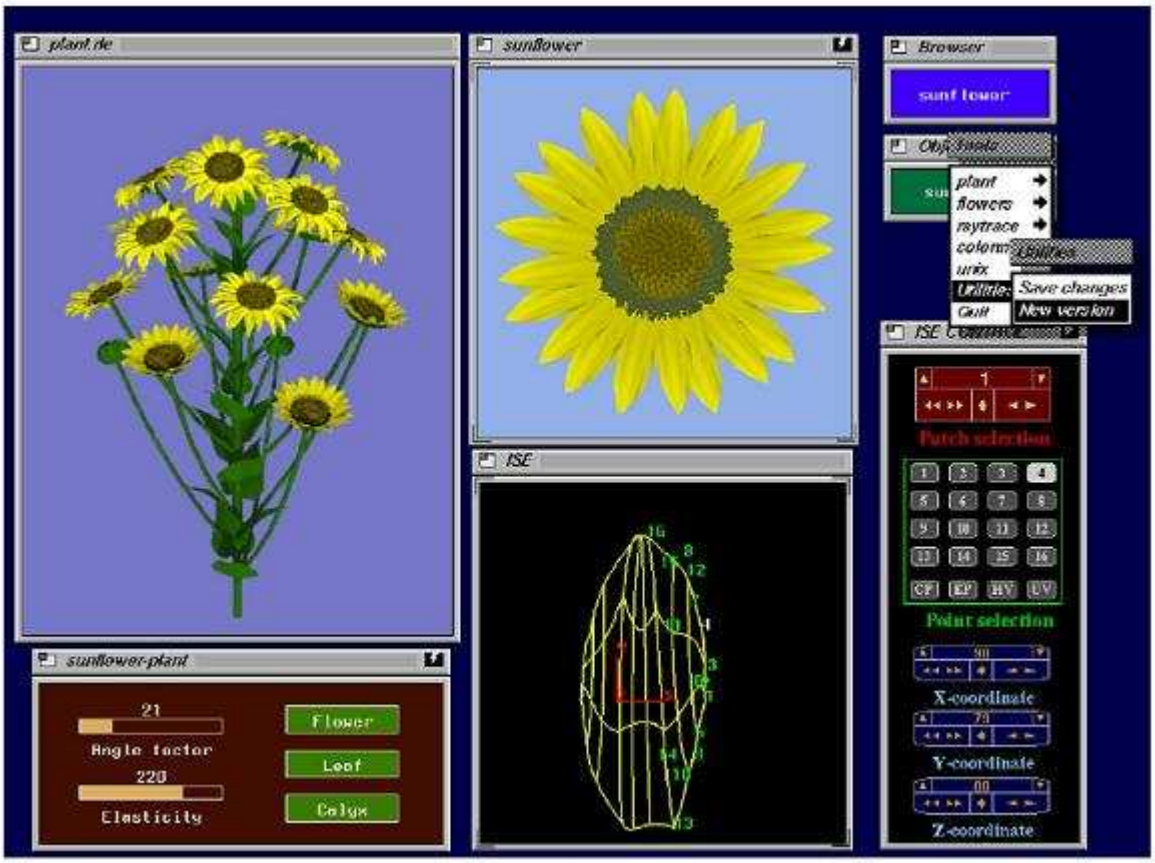


Abbildung 3.10: Das virtuelle Labor aus Algorithmic Beauty of Plants (aus [PL96])

3.3 Regelbasierte Objekterzeugung

Regelbasierte Ojekterzeugung nach [LD99], [DL97] und [Deu03] bezeichnet eine Vorgehensweise, die als Mischung aus L-Systemen und prozeduralen Methoden beschrieben werden kann. Damit möchte man die Mächtigkeit von regelbasierten Systemen mit der intuitiven Modellierung von prozeduralen Methoden verbinden. Wenn wir noch einmal das Bild 3.8 betrachten, dann werden wir sofort einsehen, dass diese Art der Modellierung wenig intuitiv ist. Grundlegend für solche Systeme ist ebenfalls eine Trennung zwischen Geometrie und Topologie (wie auch in Kapitel 3.1.4 beschrieben). Die Topologie wird durch einen Graphen beschrieben, in dem die grundlegenden Komponenten aus denen die Pflanze besteht aneinandergefügt werden, während die Geometrie der einzelnen Komponenten prozedural erzeugt wird.

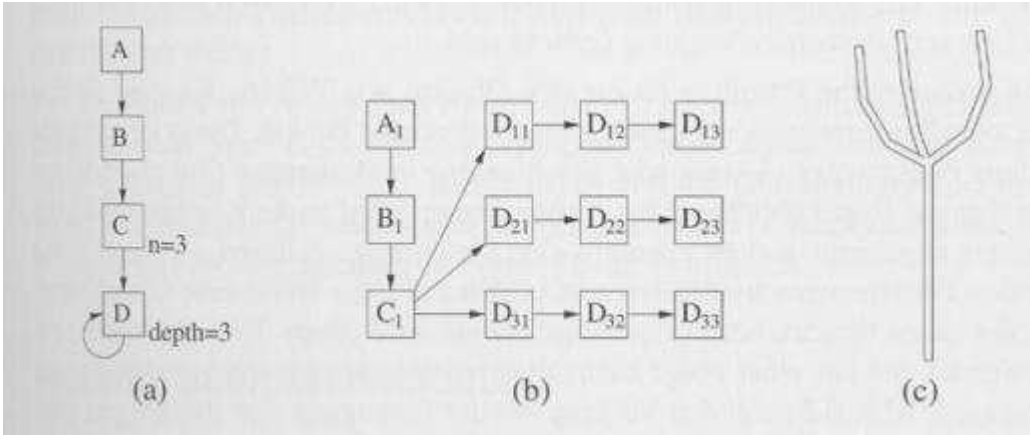


Abbildung 3.11: Vom Modellieren zur Geometrie (aus [Deu03])

In Bild 3.11 sehen wir den grundsätzlichen Aufbau beim Modellieren anhand der regelbasierten Objekterzeugung. Der Benutzer modelliert den sogenannten p-Graphen (Teilbild (a)) und gibt den darin enthaltenen Komponenten entsprechende oder gewünschte Parameterwerte. Bei der Erzeugung der Geometrie wird dieser Graph dann zuerst in einem i-Baum (Teilbild (b)) umgewandelt, indem rekursive und multiplikative Komponenten aufgelöst werden. Daraus wird dann erst die tatsächliche Geometrie erstellt.

Es gibt die folgenden Arten von Komponenten zum Einsatz im p-Graphen:

Kamera-Komponente

Dies ist die Grundkomponente des Systemes und muss immer als erste Komponente im p-Graphen stehen. Sie beherbergt alle Parameter, die für die Ansicht des Modells benötigt werden, zum Beispiel Lage und Ausrichtung der Kamera und Position und Art der Lichtquellen.

Basis-Komponente

Von dieser Komponente sind alle anderen Komponenten abgeleitet. Mithilfe der Basis-Komponente kann Geometrie erzeugt werden, als geometrische Primitive lassen sich hier sowohl Würfel, Kugeln, Zylinder oder Tori aber auch eine Menge diskreter Punkte erzeugen. Die Punktmenge kann als offene (Area) oder geschlossene (Tube) Punktmenge definiert werden und liegen üblicherweise in einer Ebene. Wird an die Komponente eine weitere angehängt, die ebenfalls eine Punktmenge definiert, dann werden die beiden Punktmengen trianguliert und bilden auf diese Weise eine Oberfläche. In der weiter unten beschriebenen Horn-Komponente wird dieser Mechanismus intern angewendet, um aus einer Folge triangulierter Punktmengen Äste und Stiele zu erzeugen.

Rotationskörper-K.

Diese Komponente erzeugt ein weiteres geometrisches Primitiv: einen Rotationskörper. Der Benutzer kann die Silhouette als polygonale Kurve editieren sowie die Auflösung in der Drehrichtung bestimmen. Da die Editierung einen speziellen Dialog nötig macht, wurde sie von der Basis-Komponente getrennt.

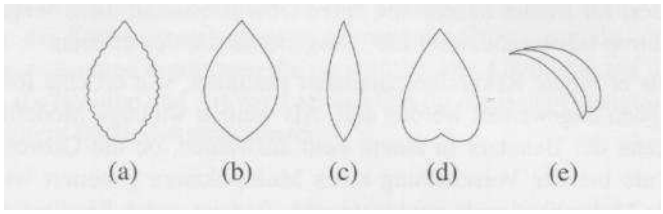


Abbildung 3.12: Verschiedene Arten von Blättern (aus [Deu03])

Blatt-Komponente

Diese Komponenten werden für alle Arten von Blättern und Blütenblättern verwendet. Die Blattoberfläche wird durch eine Folge von Area-Primitiven erzeugt, die anschliessend trianguliert werden. Verschiedene Parameter bestimmen das Aussehen der Blätter. Es gibt Parameter, die beschreiben, ob ein Blatt herzförmig gebogen wird und wie breit es ist. Die Blattoberfläche kann entlang oder quer zur Hauptachse gebogen werden und die Umrandung wird anhand einer polygonalen Kurve bestimmt bzw. vorgegeben (ein paar Beispiele in Bild 3.12). Durch den einstellbaren Fototropismus kann das Blatt unabhängig von der Stellung zum Ast dem einfallenden Licht entsprechend ausgerichtet werden.

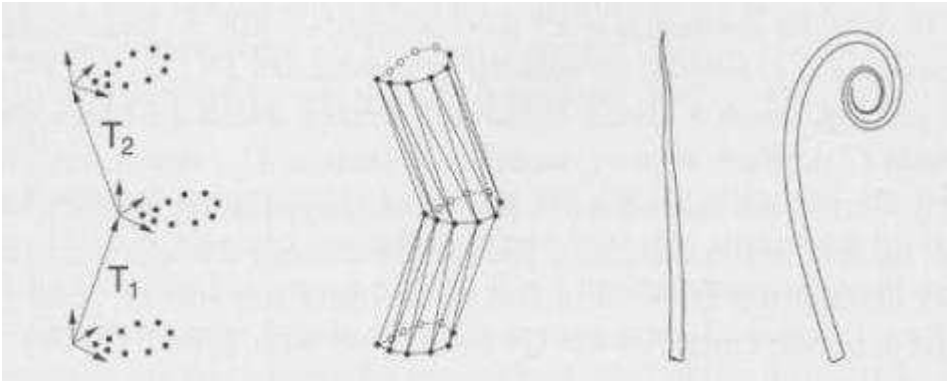


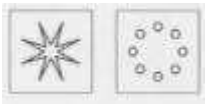
Abbildung 3.13: Aus Punktmengen entstehen Pflanzenteile (aus [Deu03])

Horn-Komponente

Die von dieser Komponente erzeugte Geometrie ist die Grundlage für alle Arten von Ästen, Stielen und Stämmen. Sie stellt im Prinzip einen generalisierten Zylinder dar, der durch die Triangulation von aufeinanderfolgenden Punktmengen, die jeweils einen Querschnitt beschreiben, (siehe auch 3.13) entsteht. Die Horn-Komponente ermöglicht es ausserdem, die Geometrie nachfolgender Komponenten zu vervielfältigen. In diesem Fall werden die lokalen Koordinatensysteme der Kindkomponenten an die Orte entlang der Kurve gehangen, die zusammen mit den Tube-Primitiven den Zylinder definieren.

Baum-Komponente

Wie bei der Horn-Komponente wird auch hier ein generalisierter Zylinder erzeugt. Der Unterschied zur Hor-Komponente liegt in der Art, wie nachfolgende Komponenten vervielfältigt werden und über welche Parameter die Form des Zylinders beeinflusst werden kann. Üblicherweise wird ein Baum als Kaskade von Baum-Komponenten aufgebaut oder über eine Rekursion definiert. Im letzten Fall können die Parameter nicht individuell für jede Verzweigungsebene eingestellt werden, was oft nachteilig ist. Die Parameter der Baum-Komponente sind in drei Gruppen unterteilt. Eine Gruppe bestimmt die Anordnung der verzweigenden Äste. Hier können die Verzweigungswinkel sowie die Verzweigungscharakteristik eingestellt werden. Diese bestimmt wie viele Äste pro Längeneinheit entlang des Stammes erzeugt werden, sie wird mittels einer Dichtekurve vom Benutzer eingestellt. Weitere Parameter bestimmen die Größe der Äste, deren Dicke entlang der Sprossachse, die Verdrehungen und Windungen des Stammes sowie die Richtungsänderungen des Stammes bei der Abzweigung eines Astes. Ausserdem kann der Verlauf des Stammes auch direkt über einen Spline definiert werden. Wie bei den Blättern können auch hier Tropismen einwirken, hierzu werden die Tube-Primitive lokal gedreht, was insgesamt eine Biegung der Geometrie erzeugt. Das Aussehen vieler Bäume ist massgeblich durch diesen Parameter bestimmt, der Effekt kann aber auch zum Modellieren der Verformungen durch Windeinwirkungen genutzt werden.

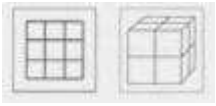


Hydra- und Wreath-K.

Die Hydra-Komponente multipliziert alle an sie angehängten Komponenten und plaziert sie in einem sternförmigen Arrangement. Der Benutzer kann hierbei die Anzahl und Größe der Nachfolger, den Öffnungswinkel des Sternes sowie Verdrehungen der Nachfolger bezüglich der Hauptachse als Wertebereich definieren. Die Wreath-Komponente ordnet ihre Nachfolger auf einem Ring an. Eingabeparameter sind dessen Durchmesser sowie die Anzahl der Nachfolger (siehe 3.14).

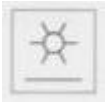
Phiball-Komponente

Werden Komponenten über eine Phiball-Komponente multipliziert, so ordnet diese die Geometrien nach der Regel des goldenen Schnittes auf einer Kugel-Oberfläche an. Der Benutzer stellt Anzahl und Größe der zu multiplizierenden Komponenten sowie den Öffnungswinkel der Kugel ein (siehe Bild 3.14).



FFD- und Hyperpatch-K.

Mit diesen Komponenten lässt sich die globale Gestalt der Pflanze beeinflussen, indem eine Freeform Defomation (FFD) definiert wird. Diese werden entweder durch vom Nutzer gegebene Funktionen $D_X(x)$, $D_Y(y)$ und $D_Z(z)$ oder über dreidimensionale Bezierfunktionen vom Grad 1 bis 3, die einen Würfel mit einer Reihe von Kontrollpunkten definieren, die vom Benutzer verschoben werden können, definiert.



Welt-Komponente

Mit dieser Komponente kann zum Beispiel das Lichtfeld durch Angabe von ortsabhängigen Funktionen für die x-, y- und z-Komponente des Richtungsvektors angegeben werden.

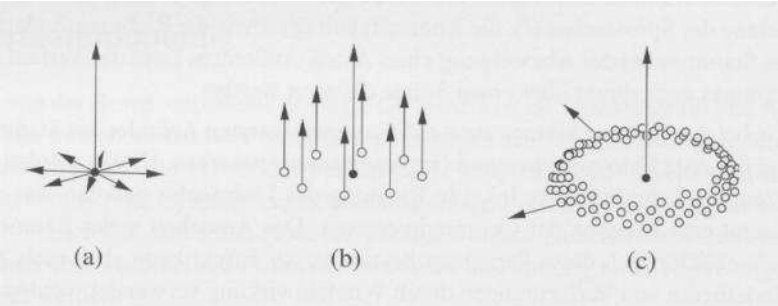


Abbildung 3.14: Hydra, Wreath und Phiball-Komponenten (aus [Deu03])

Diese Komponeten können auf die folgende Art und Weise miteinander verbunden werden:

Child-Link Hierbei handelt es sich um eine Standard-Verbindung. Die Geometrie der Komponente wird relativ zur zur Geometrie der Vorgänger-Komponente platziert. Verbindungen dieses Types werden über dünne Linien dargestellt.

Branch/Rib-Link Die Komponente wird als Verzweigung einer Baum-Komponente oder als Rippe einer Horn-Komponente multipliziert (vervielfältigt). Bei anderen Komponenten wird die Verbindung immer als Child-Link interpretiert. Die Darstellung erfolgt über dicke Kanten.

Leaf-Link Ist die Vater-Komponente Teil einer Rekursion, wird die Geometrie der Kind-Komponente nur nach Abbruch der Rekursion über den Rekursionsparameter erzeugt. Sie ist also Blatt des i-Baumes im strukturellen Sinne. Diese Verbindungen werden durch gestrichelte Linien dargestellt.

Ein einfaches Beispiel, wie eine modellierte Pflanze aussehen kann, sehen wir in Bild 3.15. Die ganzen zu den Komponenten gehörenden Parameter werden aber nicht angegeben. Dort sehen wir jeweils nur die benutzten p-Graphen und die daraus resultierenden Pflanzen.

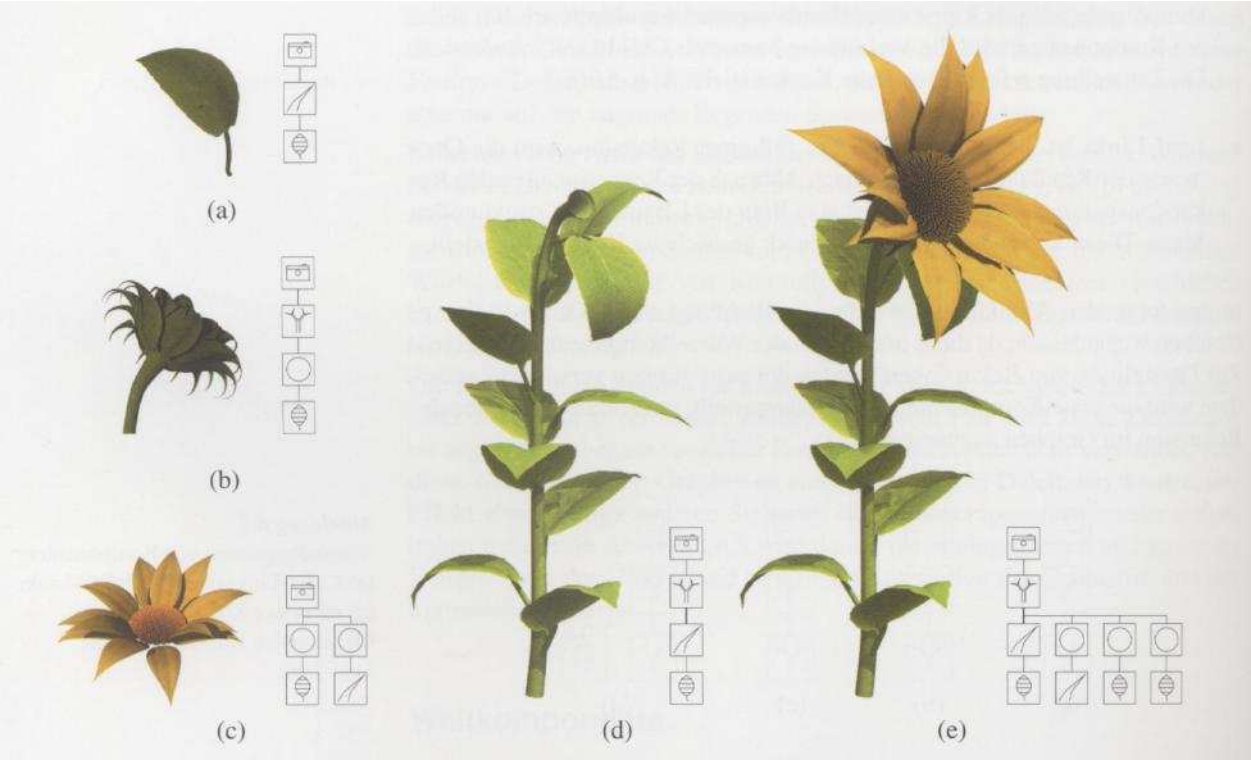


Abbildung 3.15: Eine Sonnenblume mit regelbasierter Objekterzeugung (aus [Deu03])

Kapitel 4

Modellierung von Pflanzenwachstum

In diesem Kapitel geht es im Wesentlichen um das Modellieren von Pflanzenwachstum für regelbasierte Verfahren, prozedurale Methoden werden hier nicht tiefgehend betrachtet. Unter Modellieren von Pflanzenwachstum werden hier sowohl Verfahren, die eine Pflanze und deren Aussehen besonders realitätsnah modellieren, als auch Verfahren, die schon wirklich Ansätze zur Animation des Wachstumes enthalten, verstanden.

4.1 Einfache Systeme

Es gibt verschiedene Arten, um das Wachstum einer Pflanze zu modellieren. Zu Beginn geht es noch nicht darum, die Simulation oder die Animation des Wachstumes zu modellieren, es geht lediglich darum, festzulegen an welchen Stellen und nach welchen Regeln Seitentriebe erzeugt werden. Dabei kann man Pflanzen grundsätzlich über Knospen oder über Stränge modellieren.

Ein Ansatz über Knospung (siehe [dREF+88]) simuliert das Wachstum in festen Zeitschritten. In jedem Zeitschritt kann ein Spross ruhen, absterben oder wachsen (insbesondere Verzweigungen und Blätter erzeugen). Um zu entscheiden, was genau eine Knospe in jedem Zeitschritt macht, besitzt diese mehrere Wahrscheinlichkeiten für das Ruhen, Absterben oder Wachsen.

Wie das aussehen kann, zeigt sich in Abbildung 4.1, dort ist in (a) die Wahrscheinlichkeit des Absterbens oder Aussetzens gleich null (die Pflanze wächst also in jedem Zeitschritt an jeder möglichen Position) und in (b) sind die beiden

Wahrscheinlichkeiten ungleich null, wobei abgestorbene Knospen jeweils durch x gekennzeichnet werden.

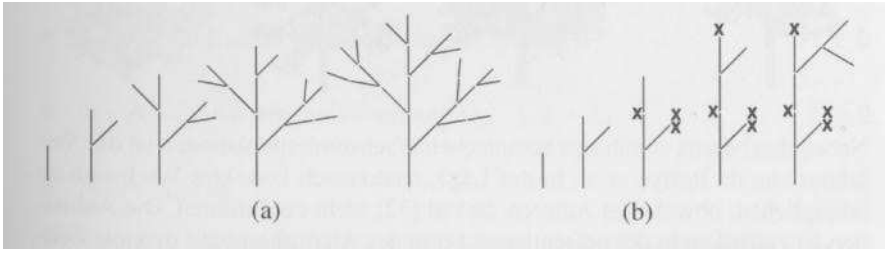


Abbildung 4.1: Ein Ansatz über Knospung (aus [dREF+88])

Neben dem Ansatz über Knospung, gibt es noch die Vermutung (die schon auf Leonardo da Vinci zurückgeht), dass ein Baum aus Strängen modelliert werden kann. In der Realität zeigt sich nämlich, dass an einer Astgabelung die Summe der Querschnitte der beiden Kinder in etwa dem Querschnitt des Vaters entspricht.

Dies wurde von Holton aufgegriffen, der seine Bäume wie in Abbildung 4.2 (a) modelliert. Die Anzahl der Stränge bestimmt hierbei nicht nur die Dicke der Äste, sondern auch deren Länge, die Anzahl der Blätter und den Verzweigungswinkel. Soll sich ein Ast mit S_0 Strängen gabeln, wobei das Verhältnis der Strangzahlen mit $P_{G,W}$ gegeben ist, so gilt für S_1 und S_2 der Kinder:

$$\begin{aligned} S_1 &= 1 + P_{G,W} * (S_0 - 2) \\ S_2 &= 1 + (1 - P_{G,W})(S_0 - 2) \\ S_0 &= S_1 + S_2 \end{aligned}$$

Den Durchmesser der Äste kann man dann mit $d_i = T * \sqrt{S_i}$ berechnen, wobei T eine Konstante ist. Die Verzweigungswinkel a_1 und a_2 werden danach wie folgt berechnet ($A_{G,W}$ ist der gesamte Verzweigungswinkel):

$$\begin{aligned} a_1 &= \frac{S_2}{S_0} * A_{G,W} \\ a_2 &= A_{G,W} - a_1 \end{aligned}$$

In Bild 4.2 (b) sehen wir, wie die Winkel bei unterschiedlich dicken Kindern aussehen. Sind beide Kinder gleich dick (rechtes Teilbild), so ergibt sich eine symmetrische Verzweigung, ist ein Kind besonders dünn, dann kann dies nur eine Abzweigung ergeben (linkes Teilbild).

Dabei sind $A_{G,W}$ und $P_{G,W}$ jeweils Funktionen, die von der Position der einzelnen Äste oder Verzweigungen (zum Beispiel der Tiefe innerhalb des Baumes) abhängen können.

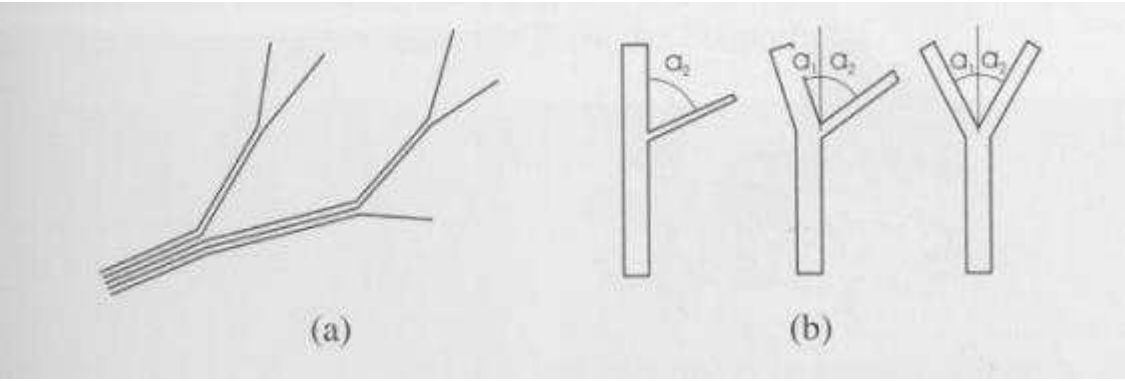


Abbildung 4.2: Ein Ansatz über Stränge (aus [Deu03])

4.2 Umwelt-Sensitive L-Systeme

In [PJM94] wird ein L-System vorgestellt, das eine Verbindung von endogenen und exogenen Bedingungen und Mechanismen beim Wachstum von Pflanzen schaffen soll.

Als allererstes wird bei diesen L-Systemen die Trennung von Anwenden der Regeln und nachträgliches Interpretieren des Ergebnisses durch beispielsweise Turtle-Grafik aufgehoben. Für die Koppelung des Wachstumes an exogene Bedingungen muss nämlich schon während des Wachstumes (also dem Anwenden von Regeln) bekannt sein, an welcher Stelle des Raumes bzw. der Umwelt ein bestimmter Teil der Pflanze sich befindet. Grundsätzlich wird ein parametriesierbares L-System verwendet, nur dass der entstehende String nach jedem Anwenden von Regeln untersucht wird und Parameter der Turtle (zum Beispiel die Position) an bestimmte in den Regeln angegebene Query-Module zur Verarbeitung gegeben werden. Diese Querymodule tauchen in den Regeln wie folgt auf: ?X(x,y,z), wobei X = P, H, U oder L sein kann und (x,y,z) je nach Wert von X entweder eine Position oder eine Orientierung sein kann (im zweidimensionalen Fall kann z weggelassen werden). Die Bedeutung der Query-Module {P,H,U,L} ergibt sich aus der Definition des entsprechendes Turtles. Dieser ist über einen Positions-Vektor $\vec{P}$ und drei Richtungsvektoren $\vec{H}$, $\vec{U}$ und $\vec{L}$ gegeben ($\vec{H}$ bezeichnet die Richtung, in die der Turtle schaut, $\vec{L}$ ist dazu im Lot der Vektor zur linken Seite, während $\vec{U}$ wiederum im Lot zu $\vec{H}$ und $\vec{L}$ den Vektor nach oben bezeichnet).

Nehmen wir als Beispiel mal ein L-System, dass eine Verzweigungsstruktur beschreibt, die auf eine Ellipse begrenzt wird:

$$
\begin{array}{lcl}
\text{w} & : & A \\
\text{p}_1 & : & A \rightarrow [+B][-B]F?P(x,y)A \\
\text{p}_2 & : & B \rightarrow F?P(x,y)B \\
\text{p}_3 & : & ?P(x,y) : 4x^2 + (y-10)^2 > 10^2 \rightarrow [+(2y)F][-(2y)F]\%
\end{array}
$$

Die erste Regel sorgt dafür, dass jeweils zwei Abzweigungen nach links und rechts erzeugt werden und die Hauptachse um eine Einheit nach vorne bewegt wird. Die zweite Regel sorgt dafür, dass die Seitentriebe geradeaus wachsen. Sobald entweder die Hauptachse oder ein Seitenspross über die Ellipse hinauswächst, greift die dritte Regel. Diese sorgt dafür, dass noch einmal zwei kurze seitliche Verzweigungen angelegt werden und dass dann durch das Symbol % das Wachstum dieses Sprosses komplett eingestellt wird. Das Ergebnis kann in Bild 4.3 betrachtet werden.

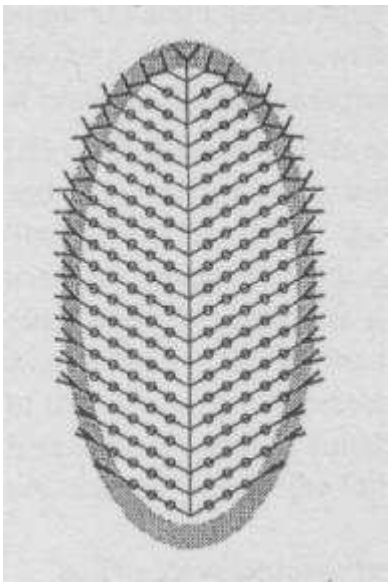


Abbildung 4.3: Wachstum auf eine Ellipse eingeschränkt (aus [PJM94])

Nun wollen wir noch ein sehr gutes Beispiel, an dem man schrittweise sehen kann, wie das Abschneiden vor sich gehen kann, betrachten.

Nehmen wir dazu die folgenden Regeln:

$$
\begin{array}{ll}
\text{w} & : \; FA?P(x,y) \\
\text{p}_1 & : \; A >?P(x,y) :!prune(x,y) \rightarrow @oF/(180)A \\
\text{p}_2 & : \; A >?P(x,y) : prune(x,y) \rightarrow \text{T}\% \\
\text{p}_3 & : \; F > T \rightarrow S \\
\text{p}_4 & : \; F > S \rightarrow SF \\
\text{p}_5 & : \; S \rightarrow \epsilon \\
\text{p}_6 & : \; @o > S \rightarrow [+FA?P(x,y)]
\end{array}
$$

Man sieht, dass dies ein kontext-sensitives L-System ist, eine Regel der Form $F > S \rightarrow SF$ bedeutet, dass ein F durch SF ersetzt wird, wenn nach dem F ein S kommt. Weiterhin steht @o für das Anlegen eines Knotens in der Verzweigungsstruktur, % ist wieder das Beenden des Abarbeitens im aktuellen Zweig, (180) steht für eine Drehung um 180 Grad anhand der Orientierung H (Heading), dies ist die Grundlage für das versetzte Wachstum der Seitenzweige, und prune ist eine wie folgt definierte Funktion:

$$prune(x,y) = (x < -L/2)||(x > L/2)||(y < 0)||(y > L)$$

Durch die Funktion prune wird also eine begrenzende Box mit der Dimension LxL definiert. Das Wachstum der Pflanze geschieht in der Produktion 1, während die Produktion 2 dafür sorgt, dass ein Ast beendet wird, sobald er über die Box hinausläuft. In Produktion 2 wird vor dem Abschneiden noch das Terminal T eingesetzt. Dieses Terminal T sorgt durch die Produktionen 3, 4 und 5 dafür, dass der letzte Teil des Astes entfernt wird. Die Produktion 6 ist für die Verzweigung beim Zurückgehen zuständig. Das Ergebnis bzw. die schrittweise Entwicklung des Baumes ist in Bild 4.4 dargestellt.

In den Beispielen in dem Papier geht es nur um das Beschneiden oder Einengen der Pflanzen auf bestimmte Räume, was allerdings auch das Fernhalten von bestimmten Räumen beinhaltet. Man kann sich aber leicht vorstellen, dass unter Ausnutzung der Orientierungsinformation der Turtle beispielweise auch so etwas wie durch Wind geprägtes Wachstum erreicht werden könnte.

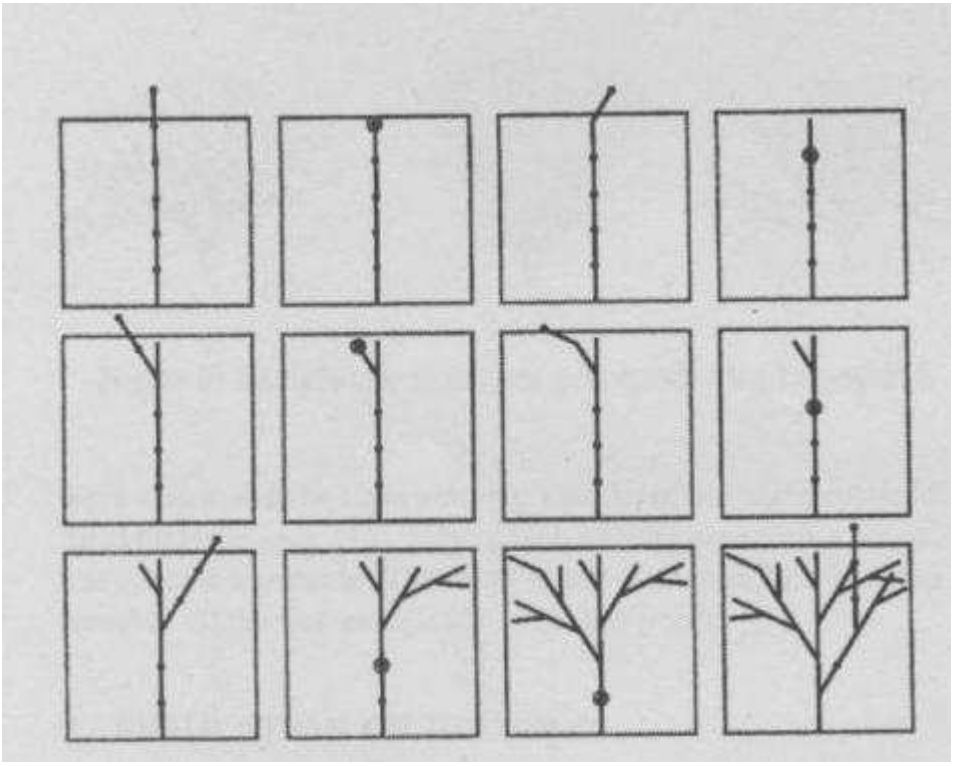


Abbildung 4.4: Schrittweise Entwicklung einer begrenzten Baumstruktur (aus [PJM94])

4.3 Offene L-Systeme

In [MP96] werden sogenannte offene L-Systeme eingeführt und als direkte Fortführung der umwelt-sensitiven L-Systeme vorgestellt. Hier geht es nicht mehr nur um die räumliche Lage oder Position von den Pflanzen (also lokale Parameter), sondern hier wird mit der Umwelt und damit auch anderen Pflanzen interagiert. Dazu wird die Umwelt neben der Pflanze modelliert und es findet ein kontinuierlicher bi-direktionaler Austausch zwischen beiden statt (zum Beispiel über herrschende Licht-Verhältnisse).

In der Kommunikation zwischen Pflanze und Umwelt werden drei Arten unterschieden:

1. Die Pflanze wird von globalen Eigenschaften der Umwelt beeinflusst. Zum Beispiel die Länge eines Tages als Signal zum Aufblühen. Oder die Temperaturen und Sonneneinstrahlung, die die Wachstumsrate der Pflanzen beeinflussen (wobei das Thema Sonneneinstrahlung eigentlich in die Kategorie 3 fällt).

2. Die Pflanze wird von lokalen Eigenschaften der Umwelt beeinflusst. Zum Beispiel ein Hindernis wie ein Stein, der das Wachstum an einer bestimmten Stelle verhindert.

3. Die Pflanze wird nicht nur von der Umwelt beeinflusst, sondern beeinflusst auch selbst die Umwelt (wobei hier unter dem Begriff Umwelt auch andere Pflanzen gemeint sein können). Als Beispiele seien hier der Wettbewerb um Raum, Nährstoffe oder Sonneneinsrahlung genannt.

Für das Modellieren dieser Interaktionen zwischen Pflanzen und ihrer Umwelt wird ein Reception → Reponse-Modell vorgeschlagen. Eine direkte Kommunikation zwischen verschiedenen Pflanzen gibt es nicht. Diese werden indirekt über die Änderungen in der Umwelt vorgenommen (siehe dazu Bild 4.5).

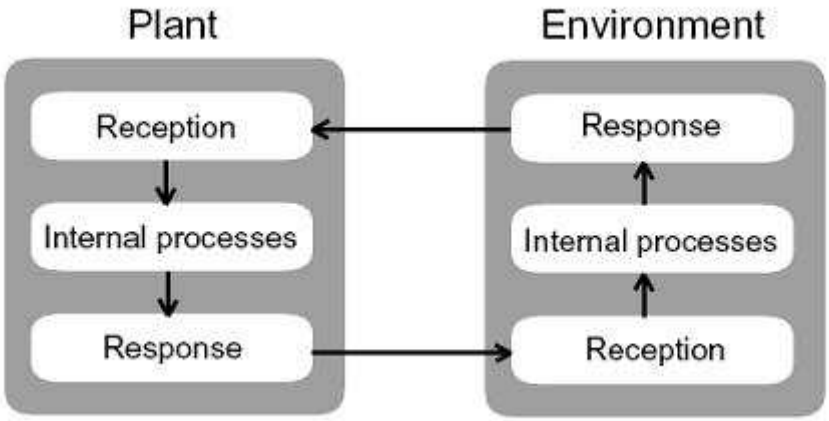


Abbildung 4.5: Kommunikation zwischen Pflanze und Umwelt (aus [MP96])

Offene L-Systeme definieren für die Kommunikation nun neue Module der Form $?E(x_1, ..., x_m)$, mit denen Informationen an die Umwelt zur dortigen Verarbeitung geschickt und danach Ergebnisse von der Umwelt erhalten werden können (siehe dazu auch Bild 4.6).

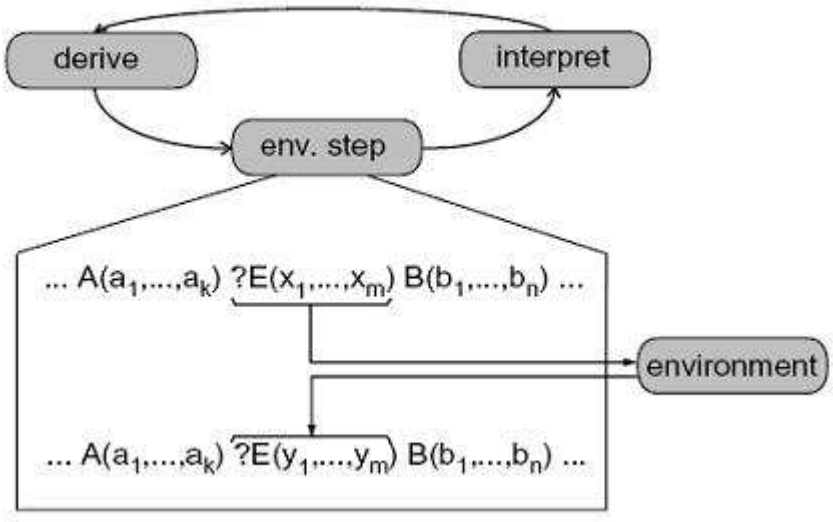


Abbildung 4.6: Die Umwelt ersetzt Parameter (aus [MP96])

Die an die Umwelt geschickten Informationen enthalten die Adresse (Position im String) des Modules. Diese ist notwendig für die Antwort der Umwelt. Neben dieser Adresse werden noch die Werte der Parameter x_i, die Position und Orientierung der Turtle, sowie der Typ und die Parameter des auf dieses Modul folgenden Modules an die Umwelt weitergegeben.

Wir wollen nun ein einfaches Beispiel über zwei Pflanzenzweige, die um Licht konkurrieren, zur Illustration des Ganzen (das End-Ergebnis sehen wir in Bild 4.7) betrachten. Das Kommunikations-Modul E hat hier nur einen Parameter x, der die Grösse eines Blattes angibt. Dieser Parameter wird zusammen mit der Position der Turtle an die Umwelt geliefert und diese entscheidet dann über das Schicksal des Zweiges. In Abhängigkeit von dieser Entscheidung (allerdings ohne Aussage, wie diese getroffen wird) wird entweder 0 oder 1 zurückgegeben. Wie man an der ersten Produktion sehen kann, wird ein Zweig sich nur dann weiterentwickeln, wenn die Rückgabe den Wert 1 liefert. Bei der Rückgabe von 0 bleibt der Zweig inaktiv. Die zweite Produktion sorgt dafür, dass nicht mehr benötigte Kommunikations-Module, also die Module, welche den Wert 0 geliefert bekamen, entfernt werden.

Dazu werden die folgenden Konstanten benutzt:

r_1	0,94	Die Rate, mit der die Länge und Grösse für den ersten Zweig abnimmt.
r_2	0,87	Die Rate, mit der die Länge und Grösse für den zweiten Zweig abnimmt.
α_1	24,4	Der Verzweigungswinkel für den ersten Zweig.
α_2	36,9	Der Verzweigungswinkel für den zweiten Zweig.
ϕ	138,5	Der Winkel zwischen den einzelnen Zweigen in der Startregel.

+, - und / sind hier wieder nur Änderungen der Winkel.

$$\text{w} : -(90)[F(1)?E(1)A(1)] + (\phi)[F(1)/?E(1)A(1)] + (\phi)[F(1)?E(1)A(1)] \\ +(\phi)[F(1)/?E(1)A(1)] + (\phi)[F(1)?E(1)A(1)]$$

$$\text{p}_1 : ?E(x) < A(y) : x == 1 \rightarrow \\ [+(\alpha_2)F(v * r_2)?E(r2)A(v * r_2)] - (\alpha_1)F(v * r_1)?E(r1)A(v * r_1)$$

$$\text{p}_2 : ?E(x) \rightarrow \epsilon$$

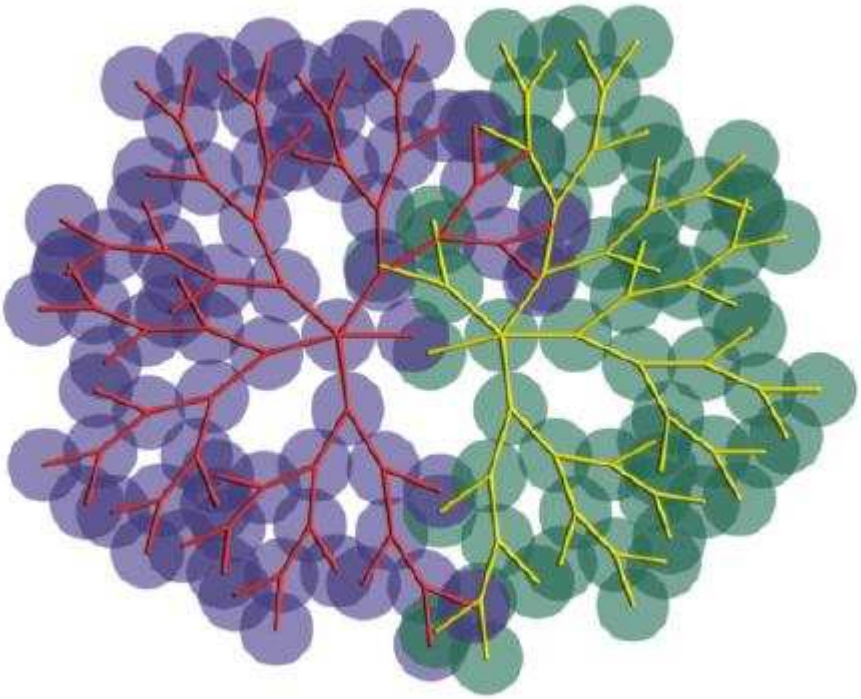


Abbildung 4.7: Zwei um Licht konkurrierende einfache Pflanzen (aus [MP96])

Insgesamt bleibt anzumerken, dass diese Vorgehensweise für eine dynamisch wachsende Pflanze nicht unbedingt zu empfehlen ist. Da in der Realität ein Stamm und Äste immer weiter wachsen, ist es nicht praktikabel, wenn zum Beispiel

Informationen über vorhandene Blätter (wegen des Schattenwurfes) an die Umwelt gemeldet werden. Diese Positionen werden sich nämlich nachträglich noch Verschieben aufgrund von Wachstumsänderungen des Stammes oder von Ästen. Eigentlich müssten an die Umwelt gemeldete Positionen daher fortwährend aktualisiert werden.

Trotzdem können mit diesem Ansatz schon interessante Ergebnisse erzielt werden, wie man zum Beispiel in Abbildung 4.8 sehen kann.

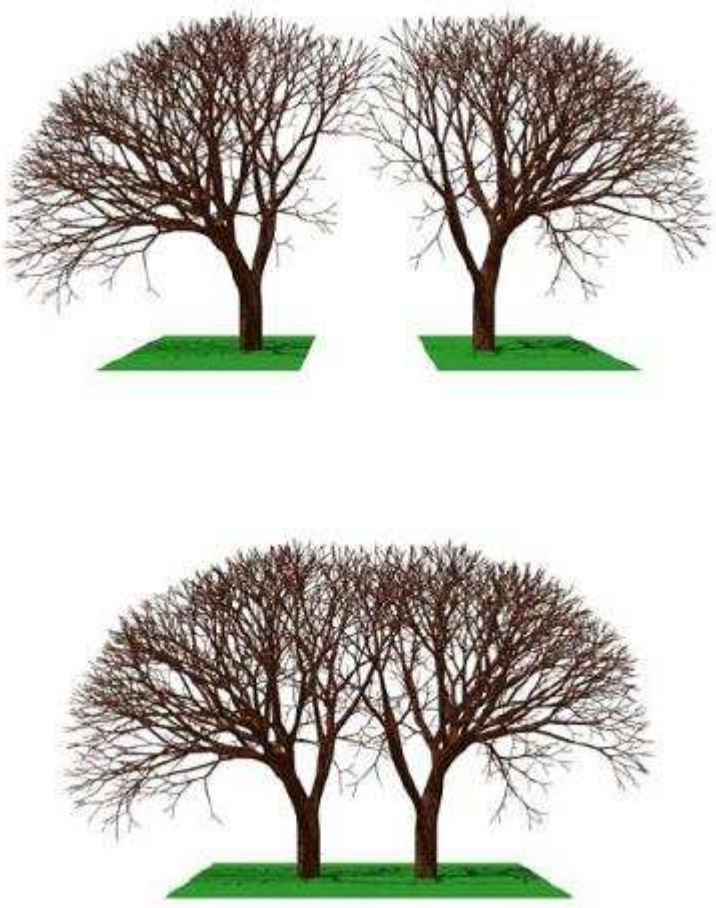


Abbildung 4.8: Zwei um Licht konkurrierende Bäume (aus [MP96])

4.4 Differentielle L-Systeme

In [PHM93] wird als erstes nicht nur der Einfluss der Umwelt auf die Wuchsform, sondern auch die Entwicklung des Wachstumes (zum Beispiel Längenwachstum über die Zeit) betrachtet. Theoretisch könnte man eine Art Längenwachstum in jedes L-System einbringen, indem man dieses wie folgt definiert:

- V={F,X}
- S={+,-}
- w= X
- P={(X → F[+X]F[-X]+X),(F → FF)}

In diesem L-System würde die erste Regel (für X) die Verzweigungen erzeugen, während die zweite Regel (für F) einem Längenwachstum entspricht. Die Anwendung dieses einfachen Mechanismus würde allerdings zu einem exponentiellen Längenwachstum der Äste führen, was in vielen Fällen nicht erwünscht ist. Tatsächlich findet man bei Pflanzen oft ein sogenanntes sigmoides Wachstum, die Entwicklungskurve lässt sich dabei als eine Art S beschreiben (Sigmoidfunktion). Zu Beginn wächst die Pflanze langsam, dann sehr schnell und gegen Ende wieder sehr langsam. Ein Beispiel für eine solche Funktion findet sich in Bild 4.9.

Insgesamt zeigt sich, dass der einfache Ansatz zum Längenwachstum sehr ungeeignet ist, die jetzt vorgestellte Erweiterung von L-Systemen verbindet daher diskrete und kontinuierliche Aspekte in der Entwicklung von Pflanzen und will damit die Animation von Pflanzenwachstum ermöglichen. Ein L-System beschreibt hier Topologie-Änderungen zu diskreten Zeitpunkten (zum Beispiel das Entstehen eines neuen Astes), während dazwischenliegende kontinuierliche Änderungen von bestimmten Parametern (Längenwachstum, Winkeländerungen, ...) durch Differentialgleichungen beschrieben werden. Dabei können die Änderungen von den Parametern der jeweiligen Nachbar-Module abhängig gemacht werden. Eine Regel des L-Systemes wird nur angewendet, wenn ein Parameter eines vorhandenen Modules aus seinem Gültigkeitsbereich hinausläuft. Welche Regel dann angewendet wird, kann davon abhängen, welcher Parameter aus seinem Gültigkeitsbereich hinausgelaufen ist.

Das Auswerten der Differential-Funktionen entspricht hier einem Anfangswert-Problem. Der Wert x für einen bestimmten Zeitpunkt t_0 ist bekannt und es geht darum, den Wert zu dem Zeitpunkt $t_0 + dt$ vorherzusagen. Ein sehr einfaches Verfahren wäre das Bestimmen der (durch die Differential-Funktion beschriebenen) Steigung m zum Zeitpunkt t_0 und die Addition des Wertes $m * dt$ zu x.

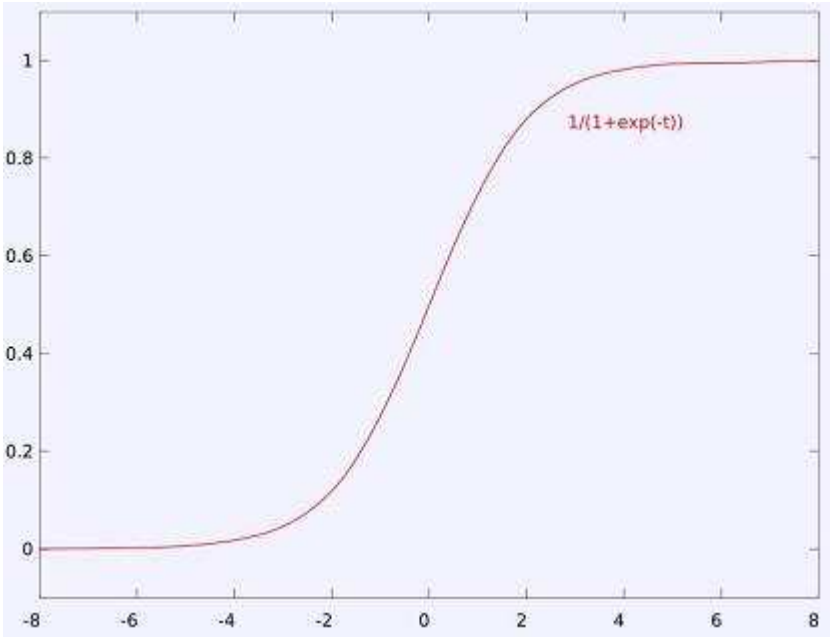


Abbildung 4.9: Beispiel für eine Sigmoid-Funktion (aus [Wik08d]

Diese Verfahren nennt man auch Euler-Cauchy. Hier geschieht das Auswerten der Differential-Funktionen allerdings mit Runge-Kutta vierter Ordnung. Runge Kutta verwendet dazu Halbschritte, bestimmt mehrere Steigungen und addiert diese dann gewichtet.

In Bild 4.10 sehen wir das Auswerten einer Differential-Funktion mit Runge Kutta vierter Ordnung einmal beispielshaft dargestellt. Die Steigung m_1 zum Zeitpunkt t_0 ist bekannt und kann benutzt werden, um den Prognose-Hilfswert Z_1 zu bestimmen als $x + m_1 * 0,5 * dt$, welcher dann den Punkt P_1 ergibt. Indem man P_1 in die Differential-Funktion einsetzt, kann eine weitere Steigung m_2 berechnet werden. Mit dieser Steigung führt man den ersten Halbschritt erneut aus und bekommt einen neuen Prognose-Hilfswert $Z_2 = x + m_2 * 0,5 * dt$. Aus Z_2 ergibt sich dann der Punkt P_2, welcher wieder in die Differential-Funktion eingesetzt werden kann, um m_3 und damit Z_3 zu berechnen. Als letztes wird nun der Punkt P_3 (welcher sich wieder aus Z_3 ergibt) in die Differential-Funktion eingesetzt, um eine vierte Hilfssteigung m_4 zu berechnen.

Diese vier Hilfssteigungen werden dann gewichtet addiert $m = \frac{1}{6} * (m_1 + 2 * m_2 + 2 * m_3 + m_4)$ und der gesuchte Wert bzw. Punkt B nun normal als $x + m * dt$ bestimmt.

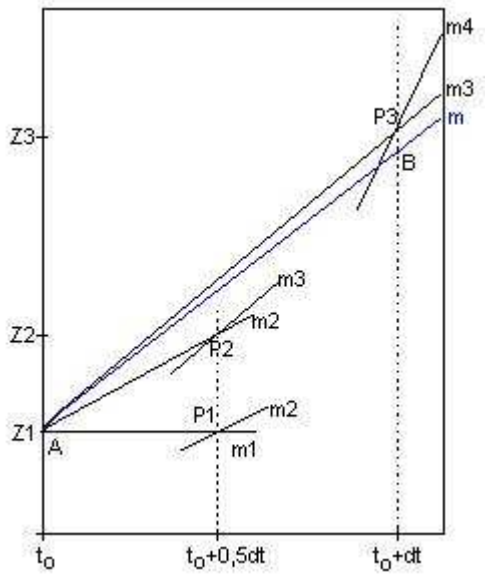


Abbildung 4.10: Auswertung einer Differential-Funktion mit Runge Kutta vierter Ordnung (aus [NRW08])

Die Animation des Gesamt-Systemes geschieht über einen Scheduler, der die Übersicht über alles hat und in gewünschten Zeitschritten Δt dafür sorgt, dass die Parameter der Pflanze aktualisiert werden. Dabei kann die Wahl des Intervalles unabhängig von den differentiellen L-Systeme geschehen. Stellt der Scheduler fest, dass ein Parameter aus seinem Gültigkeitsbereich herausgelaufen ist, dann wird eine Ersetzung im L-System durchgeführt. Stellt der Scheduler fest, dass zu einem Zeitpunkt t́ zwischen t und $t+\Delta t$ eine Ersetzung im L-System durchgeführt werden muss, so wird das Intervall in zwei Teile zerlegt und zu dem Zeitpunkt t́ noch einmal alle Parameter ausgewertet. Diese Unterteilung wird fortgeführt, falls es ein anderes Modul gibt, welches vielleicht zum Zeitpunkt t^* (zwischen t und t́ oder t́ und $t+\Delta t$) durch eine Regel des L-Systemes ersetzt werden muss. Da dies je nach Anzahl der Module und Regeln zu einem erheblichen Mehraufwand führen kann, kann man aber auch drauf verzichten, alle Module zu einem Zeitpunkt t́ auszuwerten. Dies würde im kontext-sensitiven Fall bedeuten, dass für die Nachbar-Module noch die Parameterwerte zum Zeitpunkt t benutzt werden, im kontext-freien Fall macht es gar keinen Unterschied.

Ein einfaches Beispiel für die Anwendung von differentiellen L-Systemen ist die sogenannte Drachenkurve, die aus gleichschenkligen Dreiecken mit einem rechten Winkel besteht, deren Anzahl im Laufe der Zeit zunimmt, dafür werden sie allerdings auch kleiner. Bei den einzelnen Dreiecken ist es jeweils so, dass die Hypotenuse zu Beginn ihre grösste Länge hat und dann kontinuierlich kürzer wird, während die beiden Seiten länger werden. In dem Moment, wo eine Regel des

L-Systemes angewendet werden muss, ist die Hypotenuse des alten Dreiecks verschwunden und die beiden Seiten werden als Hypotenusen für neu entstehende (und kleinere) Dreiecke verwendet.

Das dazugehörige L-System sieht wie folgt aus (- und + sind Winkeländerungen um jeweils 45 Grad, T ist die Zeit, die jede Seite braucht, um ihren grössten bzw. kleinsten Wert zu erreichen):

$F_r(x,s)$:
$if(x < s)solve\frac{dx}{dt} = \frac{s}{T}, \frac{ds}{dt} = 0$
$if(x = s)produce - F_r(0, s\frac{\sqrt{2}}{2}) + F_h(s,s) + F_l(0, s\frac{\sqrt{2}}{2}) -$

$F_l(x,s)$:
$if(x < s)solve\frac{dx}{dt} = \frac{s}{T}, \frac{ds}{dt} = 0$
$if(x = s)produce + F_r(0, s\frac{\sqrt{2}}{2}) - F_h(s,s) - F_l(0, s\frac{\sqrt{2}}{2}) +$

$F_h(x,s)$:
$if(x > 0)solve\frac{dx}{dt} = -\frac{s}{T}, \frac{ds}{dt} = 0$
$if(x = 0)produce\ \epsilon$

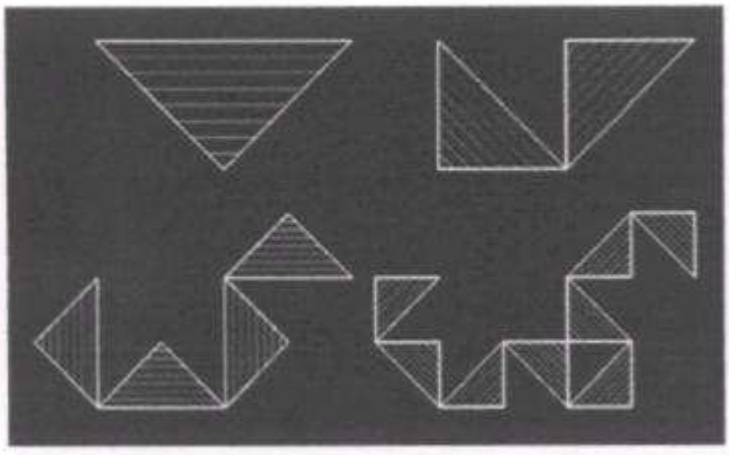


Abbildung 4.11: Entwicklung der Drachenkurve (aus [PHM93])

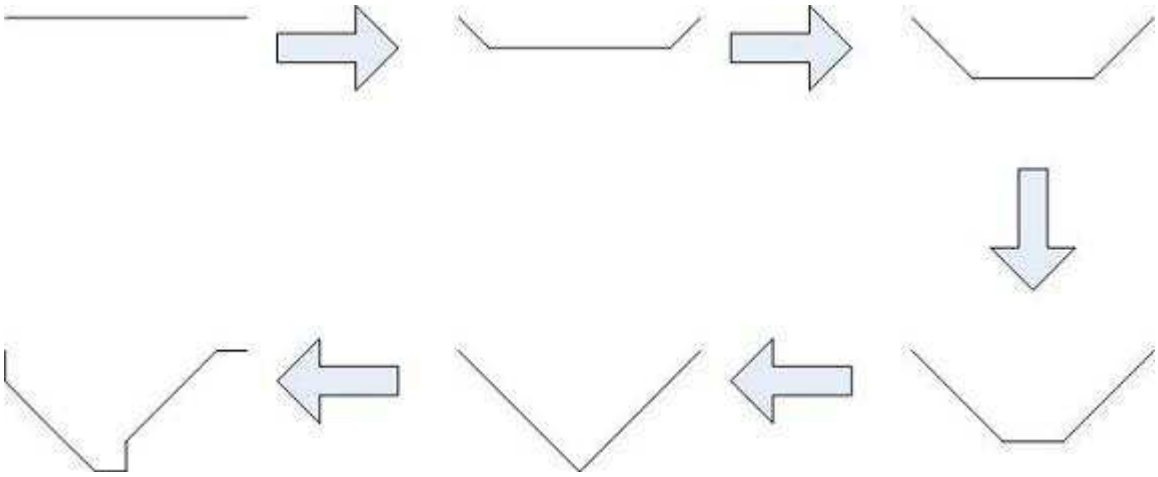


Abbildung 4.12: Entwicklung der Drachenkurve in Einzelschritten

4.5 CPFG

CPFG steht für ‘plant and fractal generator with continous parameters‘ und versucht, das mehr oder weniger starre Gerüst von L-Systemen mit dynamischen Konstrukten und dem Einarbeiten von verschiedenen vorherigen Ergebnissen aufzubrechen. Beschrieben wird es in [PHM00].

Zu den Neuerungen zählen:

- Parameter für quantitative Attribute der modellierten Struktur, die an Symbole des L-Systemes gebunden werden.
- Programmier-Konstrukte wie lokale und globale Variablen, Arrays, Input- und Output-Funktionen und flow control management.
- Modellier-Konstrukte ohne Vorbilder in Programmiersprachen: Dekomposition und Interpretations-Regeln.
- Programmier-Konstrukte für die bi-direktionale Kommunikation zwischen Pflanze und Umwelt.
- Grafische Interpretation basierend auf Turtle-Grafik.

Auf die Punkte 1, 4 und 5 wollen wir hier nicht näher eingehen, da diese entweder sehr einfach (Punkt 1) sind oder schon vorher behandelt wurden (Punkt 4 und Punkt 5). Die Grundlage dieses Systemes sind parametrisierbare L-Systeme, diese werden aber um allerlei Konstrukte angereichert.

Ein ganz einfaches System kann hier wie folgt aussehen:

1 $\#define\ growth\_rate\ 2$
2 lsystem: 1
3 derivation length: 5
4 axiom: A
5 $A - - > I(1)[+A][-A]I(1)A$
6 $I(x) - - > I(growth\_rate * x)$
7 endlsystem

Die Zeile 1 ist hierbei eine Variable, während die Zeilen 2 und 7 ein L-System einschliessen. Zeile 3 gibt die Anzahl der anzuwendenden Ableitungen wieder, Zeile 4 die Startregel und die Zeilen 5 und 6 die Produktionen. Das Symbol I steht für einen Zweig, der dazugehörige Parameter ist die Länge. Die zweite Produktion sorgt dafür, dass der Zweig in der Länge wächst (siehe in der Abbildung 4.13 Links Mitte), während die erste Produktion für das Anlegen der Verzweigungen zuständig ist (siehe in Abbildung 4.13 Links Oben). Insgesamt kann mit diesem L-System eine Struktur wie folgt modelliert werden (die Anwendungen der verschiedenen Generationen ergibt sich im folgenden Bild unten von links nach rechts):

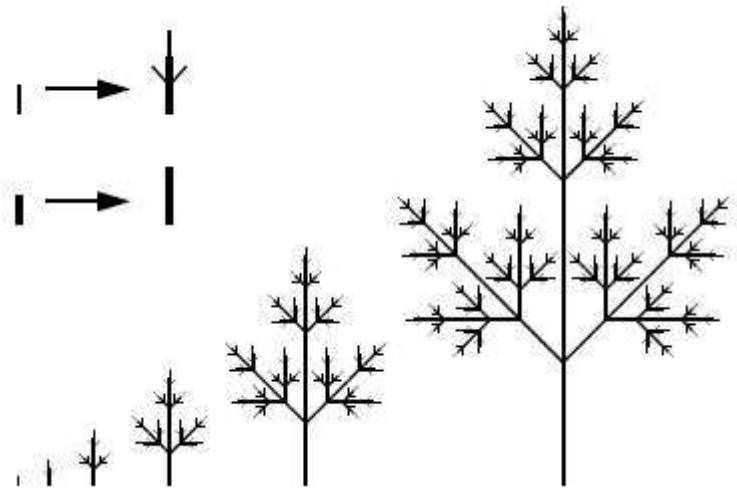


Abbildung 4.13: Einfaches Beispiel zu CPFG (aus [PHM00])

Natürlich war das nur ein sehr einfaches Beispiel für die Möglichkeiten von CPFG, grundsätzlich haben zum Beispiel die Produktionen die folgende Form:

$lc < pred > rc : \{\alpha\} cond \{\beta\} - - > succ : prob$

lc steht für den linken und rc für den rechten Kontext. $pred$ wird durch $succ$ ersetzt, wenn die Bedingung $cond$ wahr ergibt und falls die Wahrscheinlichkeit $prob$ zutrifft. α und β sind Blöcke von Anweisungen in einer C-ähnlichen Syntax, von denen α vor dem Auswerten von $cond$ ausgeführt wird und falls die Auswertung wahr ergibt und die Regel angewendet wird, dann werden die Anweisungen in β auch ausgeführt.

Ein etwas komplexeres Beispiel zu den Programmier-Konstrukten und den damit verbundenen Steuerungs-Möglichkeiten findet sich in folgendem L-System:

```
 1  #define dt 0.03
 2  #define t_max 1.0
 3  lsystem: 1
 4  Start: {fp = fopen('statistics','w');step=0;}
 5  StartEach: {step = step +1; n = 0;}
 6  EndEach: {fprintf(fp,'Step %f, number of apices %f', step, n);}
 7  End: {fclose(fp)}
 8  derivation length: 200
 9  axiom: A(0)
10  A(t) : {t_new = t + dt;} t_new < t_max {n = n + 1} - - > A(t_new)
11  A(t) : {t_new = t + dt;} t_new >= t_max
           {t_init = t_new - t_max; n = n + 3;}
           - - > I(t_init)[+A(t_init)][-A(t_init)]I(t_init)A(t_init)
12  I(t) - - > I(t + dt)
13  endlsystem
```

In diesem Beispiel wird in Zeile 4 vor dem Beginn eine Datei statistics geöffnet, in die bei jedem Schritt Informationen über die Anzahl der in diesem Schritt entstandenen Knoten reingeschrieben werden sollen. Die Anzahl der entstandenen Knoten wird in n gespeichert, welches daher in Zeile 5 immer wieder auf 0 gesetzt wird. Zeile 6 schreibt am Ende eines jeden kompletten Ersetzungsschritt die Zahl in die Datei und Zeile 7 schliesst die Datei, nachdem die gesamte Simulation durchgelaufen ist.

Neben diesen Steuerungs-Möglichkeiten werden hier noch die Konstrukte Dekomposition und Interpretation eingeführt, die wir gleich noch mit jeweils einem weiterem Beispiel vorstellen wollen. Die Dekomposition ist hier eine strukturelle Relation. Sie gibt an, wenn ein Modul eigentlich aus anderen Modulen besteht. Sie werden in einem L-System nach dem Schlüsselwort decomposition deklariert

und das Auswerten des L-Systemes geschieht dann in einem zweistufigen Verfahren. Zuerst werden die normalen Produktionen angewendet und danach die Dekompositions-Produktionen. Betrachten wir hierzu das folgende Beispiel:

1 $\#define\ dt$ 1.3
2 $\#define\ t\_max$ 1.0
3 lsystem: 1
4 derivation length: 0
5 axiom: $A(0)$
6 $A(t) - - > A(t + dt)$
7 $I(t) - - > I(t + dt)$
8 decomposition
9 $A(t) : t >= t\_max\{t\_init = t - t\_max\} - - > M(t\_init)A(t\_init)$
10 $M(t) - - > I(t)[+A(t)][-A(t)]I(t)$
11 endlsystem

Die normalen Produktionen erhöhen hier nur das Alter der Module A respektive I. Erreicht ein Modul A sein maximales Alter, dann wird es durch die Module M und A ersetzt, wobei M gleich wieder durch I und verzweigende A ersetzt wird. Im Grund ergibt sich die gleiche Struktur wie in Bild 4.13, nur dass dieses Mal die Wachstumsrate und die Zeitpunkte, zu denen Verzweigungen angelegt werden, voneinander getrennt sind. Das Anwenden der Dekompositionsregeln sieht man in Abbildung 4.14, die dicke Linie ist die Produktion des L-Systemes, die dünneren Linien sind das Anwenden der Dekomposition-Regeln.

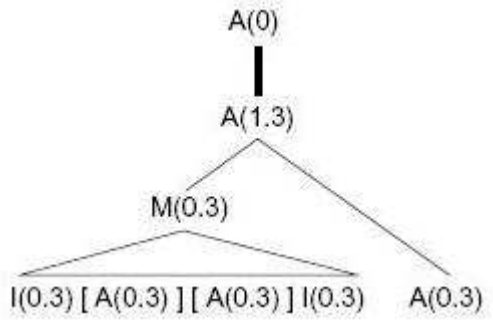


Abbildung 4.14: Anwenden von Dekompositions-Regeln (aus [PHM00])

Während die Dekomposition also konkrete Auswirkungen auf das Anwenden der Produktionen haben, sind Interpretationen unabhängig davon. Interpretationen dienen dazu, die durch das L-System beschriebene Topologie durch Geometrie-Informationen für die Visualisierung zu erweitern. Interpretationen haben aber keinen Einfluss auf das Anwenden weiterer Produktionen. Dies kann man in Bild

4.15 sehen. Durch die Interpretation h entsteht dort jeweils ein String v_i, der schon grafische Informationen enthält.

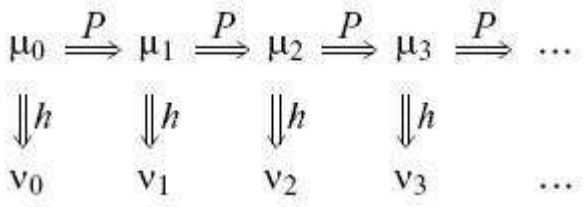


Abbildung 4.15: Anwenden von Interpretationen (aus [PHM00])

Um beispielsweise das Bild 4.13 aus dem vorhergehenden Modell (mit den Dekompositions-Regeln) zu erhalten, kann man die folgenden Interpretations-Regeln addieren:

$A(t) - - > F(t/t\_max)$
$I(t) - - > F(0.5 * 2^{(t/t\_max)})$

Hierbei werden die Symbole A und I jeweils durch F ersetzt (welches wie wir uns erinnern, bei der Interpretation durch Turtle-Grafik einer einzelnen Linie entspricht).

Interpretations-Regeln erlauben damit das Trennen der Logik des Entwicklungs-Prozesses von der Visualisierung der Pflanze.

Einführung von Unter-L-Systemen

Zur Unterstützung von strukturiertem Programmieren und um die Möglichkeit zu haben, ein grosses Modell auf mehrere und kleinere L-Systeme aufteilen zu können, werden hier Unter-L-Systeme eingeführt. Einzelne Pflanzen-Komponenten, wie zum Beispiel Blätter können damit unabhängig von der Gesamtgestalt und der Entwicklung der Gesamtpflanze in einem L-System modelliert werden und später an die richtigen Stellen gesetzt werden.

Zur Unterscheidung der verschiedenen L-Systeme wird dann das Label hinter dem Keyword lsystem benutzt und es wird eine besondere Syntax für den Sprung zwischen den verschiedenen L-Systemen eingeführt. Betrachten wir hierzu das kleine Beispiel:

1 lsystem: 1
2 derivation length: 5
3 axiom: A
4 $A - - > I[\$(2)A\$]A$
5 endlsystem

```
 6 lsystem: 2
 7 derivation length: 1 *ignored*
 8 axiom: ABC *ignored*
 9 A-->B
10 B-->C
11 endlsystem
```

Durch $(2) und $ wird der Sprung von L-System 1 in L-System 2 durchgeführt. Die Startregel von L-System 2 wird dabei ignoriert (genauso wie die Auswertungstiefe) und der Startzustand von L-System 2 wird zwischen dem $(2) und $ in L-System 1 angegeben. Das Ergebnis der Auswertungen sieht dann wie folgt aus:

A
$I[\$(2)A\$]A$
$I[\$(2)B\$]I[\$(2)A\$]A$
$I[\$(2)C\$]I[\$(2)B\$]I[\$(2)A\$]A$

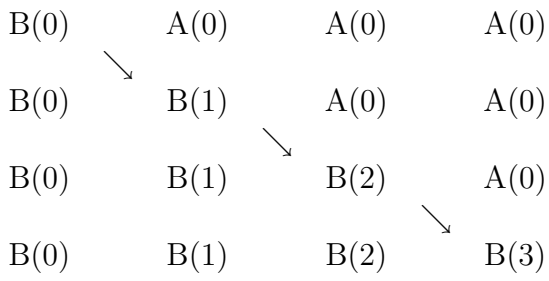
Diese Unter-L-Systeme kann man schon etwas mit den Komponenten bei der regelbasierten Objekterzeugung vergleichen und damit ist ein schon etwas intuitiveres Modellieren als noch in den anfänglichen Ansätzen üblich. Insgesamt ist CPFG leider ein Schritt in die Experten-Richtung (das heisst, dass eigentlich nur ausgewiesene Experten Pflanzen modellieren können), der durch das im folgenden Abschnitt vorgestellte Verfahren sogar noch vertieft wird.

4.6 L+C

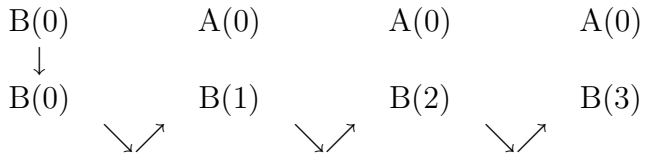
L+C (siehe [KP03]) kombiniert CPFG (siehe vorherigen Abschnitt) und Features von C++. Weil es die aktuelle Fortführung von CPFG und der Arbeit von Prusinkiewicz ist, wollen wir kurz hier darauf eingehen, um dann festzustellen, dass hier noch keine Animations-Unterstützung vorgesehen ist. Das grundlegende Augenmerk von L+C liegt jetzt auf den folgenden Punkten:

1. Schneller Informations-Transfer
2. Strukturierte und benutzerdefinierte Parameter
3. Verstärkte Kontrolle der Produktionsausführungen der L-Systeme

Um Informationen innerhalb eines L-Systemes bzw. der damit dargestellten Pflanze zu transportieren, konnte man bisher nur eine Vorgehensweise nutzen, die in jedem Ableitungs-Schritt eine Information an einen Nachbarn weitergegeben hat. Dies kann mit einer Produktion $B(m) < A(n) \rightarrow B(m+1)$ geschehen und sieht dann wie folgt aus:



L+C enthält nun ein Konstrukt, dass es erlaubt, das Ganze in einem Ableitungsschritt zu erledigen, also



Zusätzlich enthält L+C ein Konstrukt zu dem Anlegen und Benutzen von benutzerdefinierten Datenstrukturen (struct) und ein Konstrukt zu dem Ausführen von imperativen Anweisungen im C++-Stil. Diese Anweisungen können sowohl im Inneren von Produktionen angegeben und ausgeführt als auch zu weiteren definierten Zeitpunkten wie Beginn und Ende der Simulation, Beginn und Ende eines Ableitungs-Schrittes (die letzten Dinge sind schon aus CPFG im vorherigen Abschnitt bekannt) angegeben werden.

Neu ist hier auf jedenfall, dass jetzt im Inneren einer Produktion auch noch C++-Anweisungen ausgeführt werden können und dass eine Methode produce benutzt wird, um die Folge-Symbole zu erzeugen. So könnte eine Regel beispielsweise wie folgt aussehen:

```
B(m) << A(n) : {
m++;
if(m % 3 == 0) produce C(m)B(m); else produce B(m);
}
```

Ein traditionelles L-System würde hier zwei einzelne Produktionen benötigen, um den gleichen Effekt zu erzielen.

Die Kombination aus all diesen Dingen kann dazu genutzt werden, dass Informationen oder Informationsänderungen von Winkeln, Positionen und auch der Masse innerhalb der Pflanze recht schnell ausgetauscht werden können. Wobei es an dieser Stelle trotzdem verschiedene Fragen aufwirft:

1. Ist es von Seiten des Anwenders oder Pflanzen-Modellierers gewünscht, dass solche Informationen in der Grammatik stehen und damit beim Modellieren mit bearbeitet werden müssen?
2. Ist selbst diese Vereinfachung wirklich effizient, wenn eine Änderung nur eintreten kann, wenn eine Regel des L-Systemes angewendet wird?
3. Wie wird in dieses System die umwelt-getriebene Animation (siehe auch Kapitel 6) eingearbeitet?

Insbesondere die letzte Frage ist für die praktische Anwendbarkeit von Verfahren zur Modellierung und Animation von computergenerierten Pflanzen von grosser Bedeutung. Das Behandeln der umwelt-getriebenen Animation wird in den bisherigen Systemen weitestgehend ignoriert, ist aber auch wegen der zu erwartenden Komplexität der Lösung (im Bezug auf die einzusetzende und zu erwartend benötigte Rechenzeit) ein besonders wichtiger Aspekt in diesem Gebiet.

4.7 Regelbasierte Objekterzeugung

In der regelbasierten Objekterzeugung nach Deussen gibt es sowohl Ansätze für das Erzeugen eines charakteristischen Bildes einer Pflanze als auch die Darstellung bzw. Visualisierung der Wachstums-Animation.

Um ein charakteristisches Bild einer Pflanze zu erzeugen, sind verschiedene Möglichkeiten vorgesehen. Die einzelnen Bilder besitzen dann kleine Unterschiede, was einem die Möglichkeit gibt, mit der Definition eines Baumes unter Umständen gleich einen ganzen Wald definieren zu können

Basis-Komponente	In der Basis-Komponente ist an den Stellen, wo Komponenten durch Multiplikation vervielfältigt werden, vorgesehen, dass der Benutzer angeben kann, dass vielleicht nicht alle der vorgesehen Komponenten erzeugt werden.
Funktionales Modellieren	Die Werte für jede Instanz werden aus einem im Prototypen gespeicherten Wertebereich bestimmt, allerdings kann zusätzlich noch eine Zufallsfunktion darauf angewendet werden.
Tropismen	Werden sowohl bei der Ausrichtung von Blatt- und Baum-Komponenten berücksichtigt, können aber auch über die Welt-Komponente eingegeben werden. Mit einem nach unten gerichteten Gravitationsfeld lässt sich zum Beispiel eine Trauerweide besonders gut modellieren. Mit einem seitlich gerichteten Kräftefeld der Einfluss von Wind auf das Wachstum.
FFD	Mit Freiform-Deformationen kann sowohl eine ganze Pflanze als auch nur ein Teil davon verändert werden.
Beschneidungen	In der Baum-Komponente kann das Wachstum durch Angabe eines geometrischen Primitives als Begrenzungskörper gesteuert werden.

Die Animation des Pflanzenwachstumes wird durch parameterbasiertes Keyframing realisiert. Hierbei macht man sich zunutze, dass nur ein Teil der Pflanzentopologie im p-Graphen gespeichert ist. Zum Beispiel die Information darüber, wie viele Objekte von einer Multiplikationskomponente erzeugt werden, ist in ei-

nem Parameter der Komponente gespeichert. Der p-Graph kann folglich während der ganzen Animation als konstant angenommen werden. Die innerhalb einer Komponente spezifizierten Parameter werden als Parametersatz zu einem Zeitpunkt aufgefasst. Vor Begin der Animation werden zu bestimmten Zeitpunkten unterschiedliche Parametersätze (Keyframes) definiert. Die Parameterwerte für dazwischenliegende Zeitpunkte werden interpoliert

Dieser Ansatz hat allerdings bisher noch einige Nachteile, dazu gehört unter anderem die fehlende Interaktion mit der Umwelt für umwelt-getriebene Animation. Aber auch die notwendige Existenz eines Keyframes zu jeder Topologie-Änderung (i.a. Entstehen eines neuen Astes) ist ein grosses Problem. Ausserdem stellt sich die Frage, wie viele oder besser wie wenige Parameter wirklich benötigt werden und ob durch die Konzentration auf einige wenige Parameter und das besondere Betonen der Zeitachse eine sehr effiziente Animation von komplexen Pflanzen erreicht werden. Insbesondere der letzten Frage werden wir uns in dem folgenden Kapitel stellen.

4.8 Beeinflussende Faktoren für das Wachstum von Pflanzen

In Kapitel 2.1.2 haben wir den Begriff eines Tropismus schon einmal aufgegriffen. Hier wollen aber noch einmal kurz alle Faktoren, die das Wachstum einer Pflanze von aussen beeinflussen könnten, aufzählen. Dabei wollen wir uns kurz darüber Gedanken machen, inwiefern die Faktoren wirklich interessant sind.

4.8.1 Schwerkraft

Die Schwerkraft beeinflusst eine Pflanze derart, dass die Haupt-Sprossache typischerweise entgegen der Schwerkraft wächst, während Seitentriebe einen gewissen Winkel zu der Schwerkraft einnehmen.

Unabhängig von diesem allgemeinen Einfluss, müsste man bei den Seitenzweigen noch mit einberechnen, dass diese durch das Einwirken der Gravitation auch nach unten gezogen werden. Dies geschieht in Abhängigkeit von ihrer Länge, der Dicke und des Gewichtes der noch dran befindlichen Elemente.

Tatsächlich kann es in der Natur sogar vorkommen, dass einzelne Äste ohne menschliche Einwirkungen und schon bei geringem Wind abbrechen. Dies tritt vornehmlich während der Blüte- oder Frucht-Phase auf, wenn ein Ast durch zusätzliches Gewicht überbelastet wird.

4.8.2 Sonnenstand

Beim Phototropismus legt das Licht die Richtung der Bewegung fest. Die meisten Sprossachsen und viele Blattstiele sind positiv phototrop, wohingegen die Seitenzweige häufig plagiotrop sind. Die Blätter stehen meist in einem rechten Winkel zum einfallenden Licht. Ein weiteres Phänomen ist die Schattenflucht, bei der teilweise beleuchtete Blätter zur besonnten Seite hin verschoben werden. Daneben gibt es auch die umgekehrte Variante, bei der eine Pflanze vor zu großer Sonneneinstrahlung flieht.

Hierbei ist vielleicht noch eine kleine Unterscheidung interessant. Für die allgemeine Simulation einer Szene mit Pflanzen mag es interessant sein, zu jeder Sekunde abfragen zu können, wo die Sonne steht, während für das grundsätzliche Wachstum oder die Wuchsform von Pflanzen auch eine etwas ungenauere Information zu dem Sonnenstand ausreicht (vielleicht sogar ja nur eine durchschnittliche Position pro Monat).

4.8.3 Feuchtigkeit

Hydrotropismus beschreibt die Reaktion von Pflanzen auf Feuchtigkeit, durch diesen Tropismus werden hauptsächlich Wurzeln betroffen, die dann verstärkt in Richtung der feuchteren Gebiete wachsen. Damit wird dieser Faktor im Allgemeinen wohl nur wenig Einfluss auf die Animation haben. Eine Frage wäre allerdings, ob das Entwurzeln eines Baumes durch sehr starkes unterschiedlich ausgeprägtes Wachstum der Wurzeln begünstigt werden kann. Ob man also die Wurzelbildung (und damit den Hydrotropismus) zumindest als einen einzelnen Parameter zu einer Pflanze abspeichert.

Dies setzt natürlich voraus, dass die Feuchtigkeit im Boden auch mit modelliert oder erfasst wird. Wie dieses geschehen kann, werden wir in Kapitel 8.1.3 sehen.

4.8.4 Wind

Hier sei noch einmal auf die zwei unterschiedliche Ausprägungen von Wind hingewiesen, die für die das Modellieren und die Animation von Pflanzen interessant sind.

Da wäre zum einen der allgemeine Wind, der die Wuchsform einer Pflanze erheblich beeinflussen kann und der auch schon verschiedentlich in bisherigen Ansätzen berücksichtigt wurde. Viel interessanter ist aber der Wind, der eine Pflanze oder einen Baum von links nach rechts wiegt und im Extremfall auch dazu führen kann, dass ganze Äste abbrechen. Diese Art des Wind werden wir in Kapitel 6 noch genauer betrachten.

4.8.5 Katastrophen

In manchen Algorithmen wurde schon verschiedentlich ein Katastrophen-Faktor vorgesehen. Etwa die Möglichkeit, dass der eine oder andere Seitentrieb zufällig doch nicht erzeugt wird (owohl er da sein müsste), um durch Sturm (oder Gravitation) abgebrochene Äste zu modellieren. Für den Fall, dass die Pflanze aber schon unter Zuhilfenahme von Zufallsfunktionen und Wahrscheinlichkeiten modelliert wird, kann man diesen Katastrophen-Faktor aber wohl vernachlässigen.

4.9 Zusammenfassung

Die Zukunft des Modellieren und der Animation von Pflanzen liegt in der regelbasierten Objekterzeugung. Neuere Entwicklungen in den rein regelbasierten Systemen wie CPFG und L+C gehen auch schon etwas in die Richtung. So kann man die Unter-L-Systeme mit den Komponenten der regelbasierten Objekterzeugung vergleichen. Der Vorteil von rein regelbasierten Verfahren liegt in der Mächtigkeit des Verfahrens und in der Möglichkeit, Interaktionen zwischen Pflanzen und Umwelt und Prozesse innerhalb der Pflanzen, mitzumodellieren.

Was aber sowohl L+C als auch der regelbasierten Objekterzeugung bisher fehlt, ist die Komponente für die umwelt-getriebene Animation. Diese zeichnet sich durch üblicherweise sehr hohen Rechenaufwand aus, wobei eine Vermutung an dieser Stelle ist, dass diese sich durch das modulare Konzept der regelbasierten Objekterzeugung besonders effizient in diese integriert werden kann (im Gegensatz zu den rein regelbasierten Verfahren). Ausserdem kann es aus Sicht der Anwender und Modellierer sicher auch ein Vorteil sein, wenn es möglich ist, sich auf wenige aber dafür wirklich wichtige Parameter zu beschränken. Auch aus dieser Motivation heraus entstand der Plantanimator, der mit all seinen Möglichkeiten und Fortführungen in den nächsten Kapiteln beschrieben wird.

Kapitel 5

PlantAnimator

Der PlantAnimator entstand aus der Idee heraus, die Animation von Pflanzen noch besser (benutzerfreundlicher und intuitiver) modellieren sowie die Umsetzung möglichst effizient realisieren zu können. Die Grundidee war es, ähnlich wie in den differentiellen L-Systemen, einfache Aspekte der Animation mit in die Grammatik einzubeziehen, allerdings das Ergebnis und die Möglichkeiten deutlich zu verbessern. Aber auch gleichzeitig die Grundprinzipien der regelbasierten Objekterzeugung nicht aus den Augen zu verlieren, da diese für die Zukunft die besten Aussichten versprechen, sowie von Anfang an auf die Effizienz des Verfahrens zu achten und weitere Optimierungsmöglichkeiten wie die Speicherplatzanforderungen im Rahmen eines Ökosystemes mit vielen tausend Pflanzen zu berücksichtigen.

Um das Modellieren auch für Nicht-Experten und eine effizientere Auswertung der Funktionen zu ermöglichen, haben wir uns zusätzlich entschlossen, keine Differential-Funktionen zu benutzen. Stattdessen behandeln wir die Entwicklung der für die Animation wichtigen Parameter durch die Angabe von Anfangswert, Endwert und einem Funktions-Alias, welches angibt wie der Wert der Funktion sich zwischen dem Anfangs- und Endwert ungefähr entwickelt. Gleichzeitig haben wir versucht, uns auf möglichst wenige Parameter zu beschränken und damit die für die Animation wichtigen Parameter zu identifizieren.

Die sogenannte umwelt-getriebene Animation (siehe auch später in Kapitel 6) wird hier allerdings noch nicht berücksichtigt.

5.1 Aufbau

Eigentlich besteht der PlantAnimator aus drei verschiedenen Editoren, die zusammen das Modellieren einer Pflanze ermöglichen, und der Visualisierungsumgebung.

- Modul-Editor
- Funktions-Editor
- Grammatik-Editor

Das genaue Zusammenspiel dieser Editoren werden wir uns im Folgenden anschauen und danach an einem konkreten Beispiel zeigen, wie das System insgesamt funktioniert.

5.1.1 Modul-Editor

Mit Hilfe dieses Editors kann ein Modul und damit eine Komponente oder ein Teil einer Pflanze modelliert werden. Einem Modul werden dabei einige grundlegende Werte und Geometrie-Informationen wie das Aussehen und die Anknüpfpunkte zugeordnet. Die Lage der einzelnen Komponenten zueinander wird dann in der Grammatik festgelegt. Dort wird nicht nur festgelegt, welche Komponente auf welche andere Komponente folgt, sondern auch das Benutzen der einzelnen Anknüpfpunkte der verschiedenen Komponenten.

Modul-Name	Der Name, über den dieses Modul im System und auch aus dem Grammatik-Editor heraus angesprochen wird.
Life-Time	Die Zeit, die dieses Modul zum Wachsen benötigt. Danach kann es entweder sterben, weiterexistieren oder ersetzt werden.
Mortality	Legt fest, ob die Komponente nach Ablauf der Life-Time abstirbt, ersetzt wird, einfach so verschwindet oder bestehen bleibt. Wobei das Absterben noch nicht unterstützt wird.
Breeding Interval	Gibt das Interval innerhalb der Life-Time an, in dem die Kinder dieses Modules erzeugt werden können.

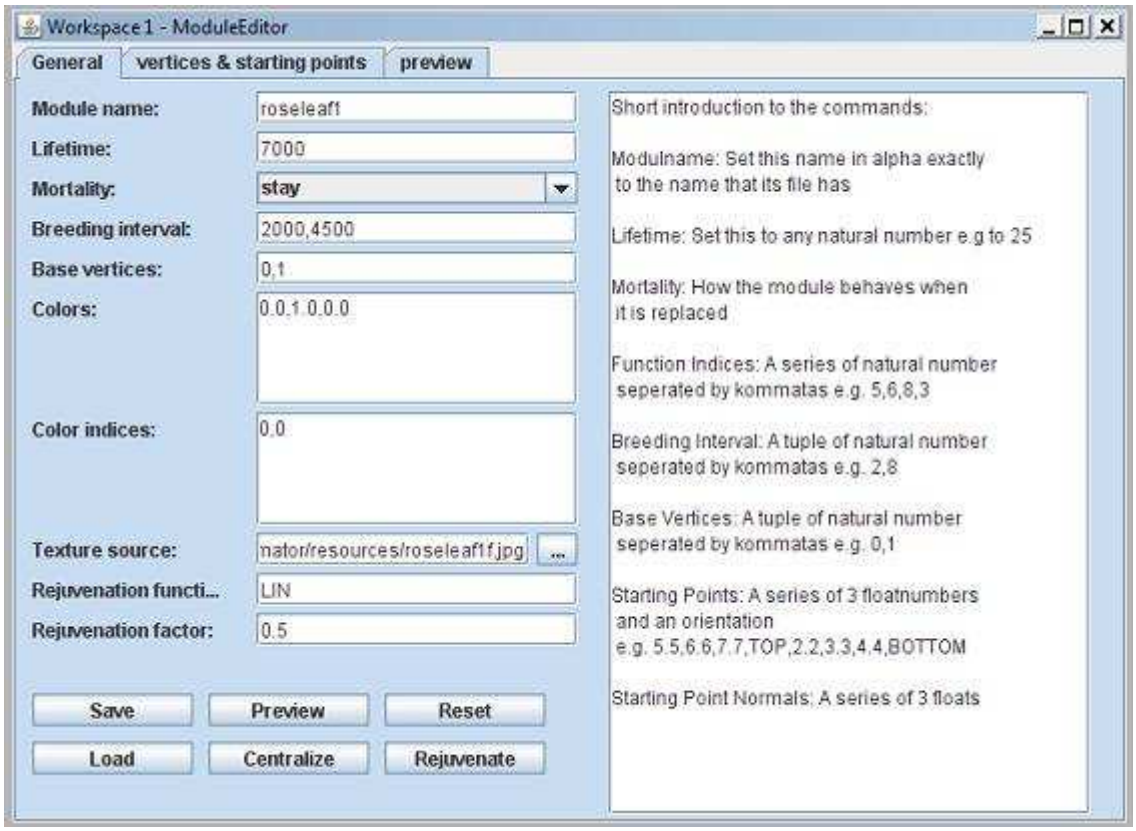


Abbildung 5.1: Der Modul-Editor, grundlegende Werte eines Modules

Base Vertices — Enthält die Indizes von zwei Vertices. Der eine ist die Basis des Modules (also die Stelle, die an einen Anknüpfpunkt des Vater-Modules gesetzt wird), der zweite wird benutzt, um mit dem ersten zusammen einen Vektor zu bestimmen, der die Ausrichtung dieses Modules beschreibt. Die Ausrichtung wird ebenfalls bei dem Positionieren des Elementes in der konkreten Pflanze benutzt, um die notwendige Translation und Rotation bestimmen zu können, damit die Ausrichtung des Modules der Ausrichtung des benutzten Anknüpfpunktes entspricht. Von dieser Ausrichtung kann allerdings in Abhängigkeit eines in der Grammatik angegebenen und zeitlich veränderbaren Winkels abgewichen werden.

Colors — Enthält die benutzten Farben jeweils als Tripel von Werten.

Color Indices — Falls keine Texturen benutzt werden, dann kann hier eine Zuordnung von Farbwerten zu Geometrie-Indizes vorgenommen werden.

Texture Source — Gibt an, welche Texture bzw. welches Bild als Texture benutzt werden soll.

Rejuvenation Function	Benutzte Verjüngungsfunktion. Die Breite eines Pflanzen-Elementes ist im Normalfall am Ende geringer als zu Beginn. Wieviel weniger breit gibt der folgende Faktor an, dieses Funktionsalias (zum Beispiel LIN für Linear) gibt an, wie sich die Änderung der Breite in Bezug auf die Länge des Pflanzen-Objektes verhält (also wie schnell die Breite abnimmt).
Rejuvenation Factor	Gibt an, um wieviel sich die Anfangs- und End-Breite dieses Pflanzen-Objektes unterscheiden.

Neben diesen grundlegenden Eigenschaften sind natürlich die Geometrie-Informationen interessant. Diese werden als Folge von Gleitpunkt-Werten beschrieben, von denen jeweils 3 Werte als x-, y- und z-Wert eines Vertices benutzt werden. Aus den Vertices bzw. den Indizes dieser Vertices lässt sich dann ein Dreiecks-Mesh aufbauen. Eine eigenständige Erstellung einer solchen Geometrie ist im PlantAnimator nicht vorgesehen, es erwies sich als praktikabel, wenn man die Geometrie in einem anderen Programm (zum Beispiel 3ds Max) modelliert und dann nur die Vertex-Informationen als Text kopiert.

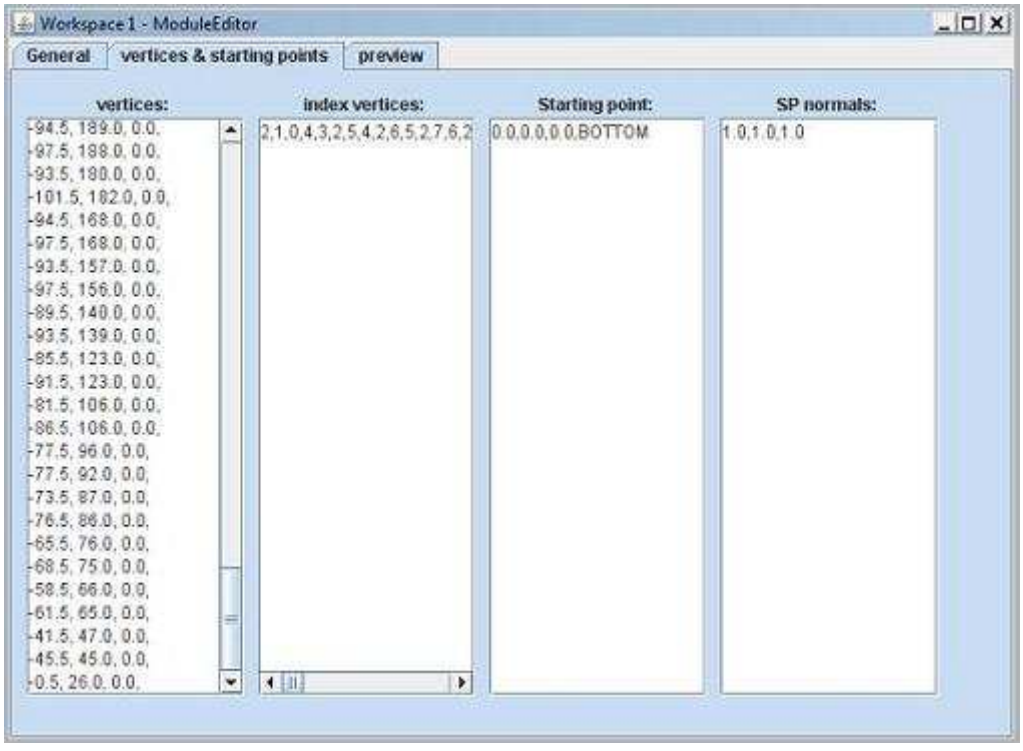


Abbildung 5.2: Der Modul-Editor, Geometrie-Informationen und Anknüpfpunkte

Zusätzlich zu der Geometrie werden noch sogenannte Anknüpfpunkte definiert. Ein Anknüpfpunkt ist dabei die Definition einer Position an der die Pflanze möglicherweise fortgeführt wird. Diese Fortführung kann entweder eine Verlängerung des aktuellen Elementes oder eine neuen Verzweigung sein. Neben der Punkt-Information wird einem Anknüpfpunkt noch ein Normalenvektor mitgegeben, der die Ausrichtung des dort beginnenden Elementes beschreibt. Im Rahmen der konkreten Visualisierung wird dann der zu einem Modul gehörende Ausrichtungsvektor (siehe Base Vertices) in Verbindung mit dem zum Anknüpfpunkt gehörenden Normalenvektor benutzt, um die Ausrichtung sowie die nötige Rotation für das Plazieren des Modules innerhalb der konkreten Pflanze bestimmen zu können. Wie die Anknüpfpunkte für den Anwender benutzt werden, sehen wir bei dem Grammatik-Editor, hier nur die Anmerkung, dass diese noch in die Klassen TOP, CENTER und BOTTOM unterteilt werden. Diese Klassen werden in der Grammatik benutzt, um festzulegen, wo die Kind-Elemente entstehen können. Damit kann verhindert werden, dass ein Seitenzweig als Fortsetzung des Haupt-Sprosses entsteht.

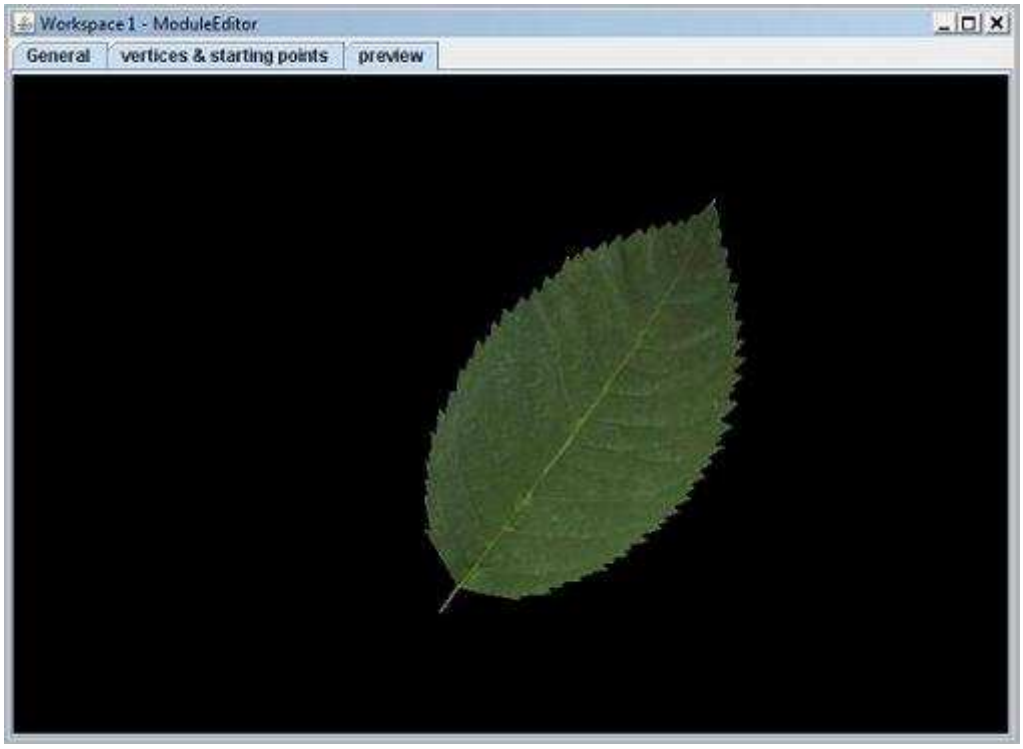


Abbildung 5.3: Der Modul-Editor, Vorschau des Modules

In dem ersten Ansatz des PlantAnimators wurden allerdings noch nicht alle genannten Parameter auch umgesetzt. Zu den nicht umgesetzten Parametern gehört die Rejuvenation aber auch das mögliche Absterben von Pflanzen-Elementen nach

dem Ende ihrer Lebenszeit. Wobei bisher noch gar nicht spezifiziert ist, wie das Absterben genau aussieht. In der Praxis würde man wahrscheinlich das Absterben auch durch ein Ersetzen mit einem dann absterbenden Modul realisieren, von daher ist eine Unterscheidung hier wahrscheinlich überflüssig. Ausserdem wird sich gleich noch zeigen, dass die Verankerung der Zeiträume, wie zum Beispiel die Lebenszeit, in den Modulen auch nicht die beste Lösung ist. Dadurch wird nämlich die Wiederverwendbarkeit der Module erheblich gesenkt.

5.1.2 Funktions-Editor

Wie oben schon beschrieben, werden die Änderungen der für die Animation wichtigen Parameter über die Kombination aus Anfangswert, Endwert und einem Funktionsalias, welches den ungefähren Verlauf zwischen den beiden erstgenannten Werten beschreibt, angegeben.

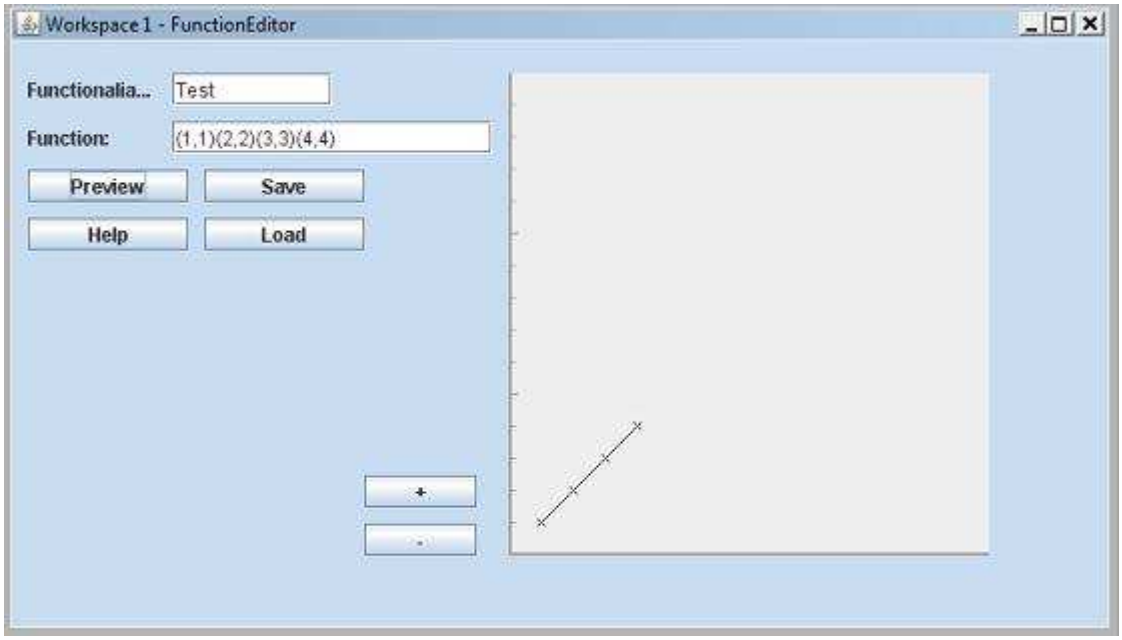


Abbildung 5.4: Der Funktions-Editor, Eingabe der ersten Punkte

Um ein Funktionsalias benutzen zu können, muss man erst eine Funktion in den Funktions-Editor eingeben. Einfache Funktionsaliase wie LIN für Lineare Entwicklung und LOG für Logarithmische Entwicklung sind allerdings bereits vorgegeben, der Benutzer kann sich aber auch ganz eigene Kurven definieren.

Zur Implementierung des ganzen Konzeptes gibt es ein Interface Function:

```
1 public interface Function {
2         double getValue(double t);
3 }
```

Im einfachsten Fall für die Implementierung der Linearen Entwicklung eines Parameters sieht der Source-Code wie folgt aus:

```
1  public class Linear implements Function {
2          double start,end,time,m;
3
4          public Linear(double start, double end, double time){
5                  this.start=start;
6                  this.end=end;
7                  this.time=time;
8                  m = (end-start)/time;
9          }
10
11         public double getValue(double time) {
12                 if(time < 0)
13                         time=0;
14                 if(time > this.time)
15                         time=this.time;
16                 return start + time*m;
17         }
18 }
```

Die Berechnung, um den Logarithmus bzw. etwas Logarithmus-ähnliches auf das entsprechende Intervall anzupassen, sieht wie folgt aus (t steht für die aktuelle Zeit, time für die Gesamtzeit der Parameterentwicklung, start und end sind die Start- und End-Werte für die Entwicklung):

```
1 public double getValue(double t) {
2         double value = Math.log(t/1000+1);
3         value += start;
4         double dt = t /time;
5         dt = Math.sqrt(dt);
6         value += dt * (end - start - Math.log(time/1000 + 1));
7         return value;
8 }
```

Gilt t == time, so hat dt auch nach dem Wurzel-Ziehen den Wert 1 und in der Berechnung von value hebt sich alles bis auf end auf. Gilt wiederum, dass t == 0 ist, so bleibt in value nur start übrig. Für den Bereich dazwischen wird durch die Benutzung der Wurzel für dt das Aussehen eines Logarithmus imitiert.

Zum Definieren von eigenen Kurven reicht es, wenn der Benutzer jeweils ein paar Punkte angibt, die Kurve wird dann mit mit Hilfe von kubischer Spline-Interpolation berechnet, dem Benutzer angezeigt und kann im Nachhinein noch intuitiv manipuliert werden. Das Angeben der Punkte (mindestens drei werden benötigt) geschieht zuerst in dem Feld Function und kann eine der beiden folgenden Strukturen haben:

$(x_0, y_0)(x_1, y_1)(x_2, y_2)$ oder $(x_0/y_0)(x_1/y_1)(x_2/y_2)$

Nachdem man die ersten Punkte angegeben hat und auf Preview drückt, werden diese Punkte auf der rechten Zeichenfläche angezeigt (es empfiehlt sich die Werte für die Punkte nicht zu gross zu wählen, damit eine gewisse Übersicht gewahrt bleibt und weil die Funktion letztendlich wie etwas weiter oben gesehen sowieso auf die für die Animation benötigten Bereiche skaliert werden muss). Nun kann der Benutzer die Punkte auf der Zeichenfläche mit der Maus verschieben und dadurch das Gesamtbild der Kurve seinen Wünschen anpassen. Um jetzt beispielsweise eine eigene Sigmoid-Kurve (siehe auch Abbildung 4.9 auf Seite 46) einzugeben, könnte man zuerst die folgenden Punkte angeben (siehe Bild 5.4)

(1,1)(2,2)(3,3)(4,4)

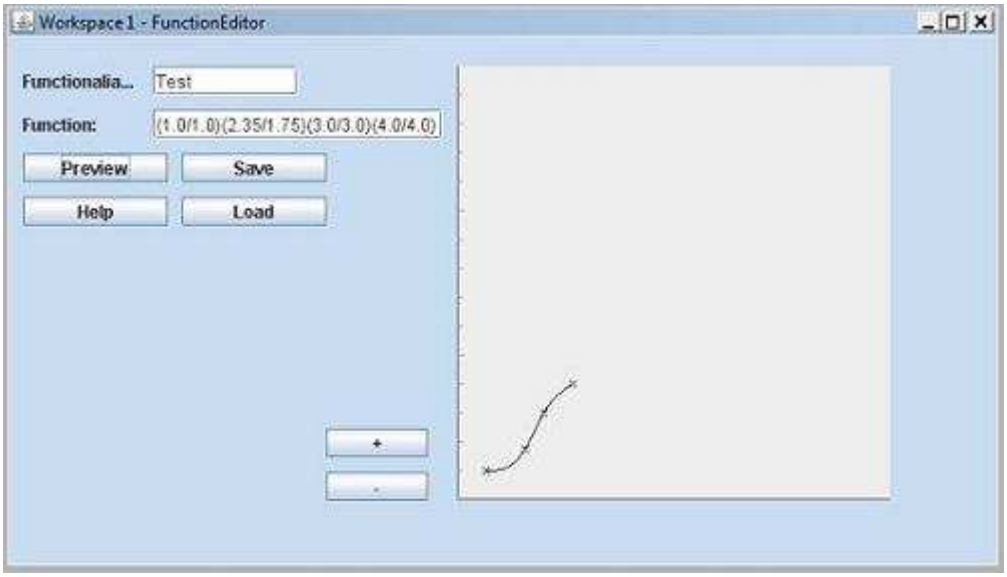


Abbildung 5.5: Der Funktions-Editor, Verschieben des zweiten Punktes

Wenn danach noch der zweite Punkt von (2/2) etwas verschoben wird (zum Beispiel auf (2.35,1.75) wie in Bild 5.5) und der vierte Punkt ebenfalls etwas verschoben wird (von (4,4) auf (4.75,4.05)), dann kann man eine Funktion wie in Bild 5.6 angegeben bekommen.

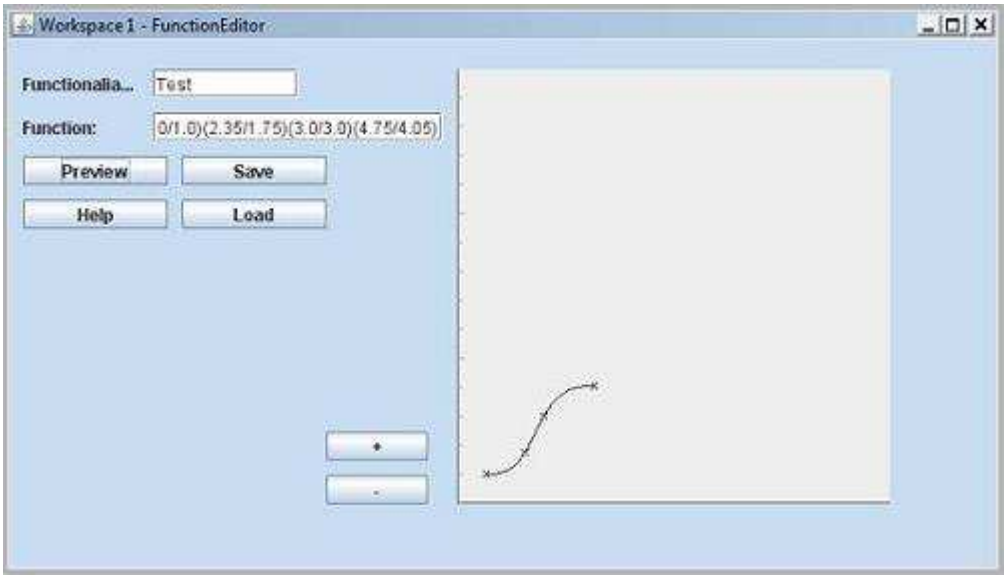


Abbildung 5.6: Der Funktions-Editor, Verschieben des vierten Punktes

Auf der Zeichenfläche können die Punkte aber nicht nur verschoben werden, es können sogar durch einen Doppelklick ganz neue Punkte eingefügt werden. Sowohl das Verschieben als auch das Einfügen neuer Punkte funktioniert aber nur innerhalb eines gewissen Gültigkeitsbereiches. So wird verhindert, dass die Funktion zum Beispiel in das Negative ausschlägt.

Das Grundproblem ist hier die Konstruktion von Kurvenstücken $s_i(x) = a_i(x - x_i) + b_i(x - x_i) + c_i(x - x_i) + d_i$ mit x zwischen zwei Stützstellen x_i und x_{i+1}, so dass der Übergang von s_i zu s_{i+1} glatt ist.

Dazu müssen die a_i, b_i, c_i und d_i in den oben beschriebenen Kurvenstücken berechnet werden, wie dies genau geschieht kann in [Sch97] nachgelesen werden.

5.1.3 Grammatik-Editor

Der Grammatik-Editor bietet die Möglichkeit, die textbasierte Grammatik einzugeben und abzuspeichern. Die Regeln der Grammatrik haben dabei die folgende Form (M_j, M_i und M_v sind hier Modulnamen aus dem Modul-Editor):

$M_j \rightarrow$ <m,k>M_i[$[a_1, a_2, F_a]$;<b_1,b_2>;Ap;Rej;$[w_1, w_2, F_w]$;$[h_1, h_2, F_h]$] + <n,j> M_v[...]

Allgemeine Konstrukte:
<x,y> : Zufallszahl zwischen x und y
$[x, y, z]$: Beschreibt jeweils eine Parameteränderung mit Startwert x,
Endwert y und Funktionsalias z

Konkrete Elemente der oberen Beispiels-Regel:
<m,k>: Zufallszahl für die Anzahl der erzeugten Pflanzenelemente
$[a_1, a_2, F_a]$: Winkel zwischen zwei Pflanzenelementen
$[w_1, w_2, F_w]$: Breite dieses Pflanzenelementes
$[h_1, h_2, F_h]$: Länge dieses Pflanzenelementes
<b_1,b_2 >: Drehwinkel in Bezug auf das Vorgängerelement M_j
Ap: Anknüpfpunkt (BOTTOM, TOP, CENTER, ALL, Benutzung von NOT erlaubt)
Rej: Verjüngungsfaktor: Um wieviel soll sich die Breite am Ende eines
Pflanzenelementes von der Anfangsbreite unterscheiden

Man sieht, dass hier schon einige Zufalls-Aspekte mit in die Grammatik eingearbeitet worden sind, es kann sogar ein zufälliger Wert für den Endwert eines Parameters eingegeben werden (zum Beispiel $< 70, 90 >$ für a_2). Der Winkel zwischen zwei Pflanzen-Elementen gibt hier den Winkel zwischen der Ausrichtung des benutzten Anknüpfpunktes und der Ausrichtung des Vater-Elementes an, mithilfe dieses Winkels kann zum Beispiel das Öffnen einer Blüte simuliert werden.

Es gibt die Möglichkeit anstatt einem Modulnamen gleich eine weitere schon vorhandene Grammatik anzugeben, diese wird dann als Teil der vorliegenden Grammatik benutzen, die Wiederverwendbarkeit beschränkt sich also nicht nur auf einzelne Module.

Als kleines Beispiel wollen wir hier die Definition einer Rosenblüte betrachten:

$S \rightarrow$
$< 1,1 > rosebudbottom1[[< 0.0,0.0 >, < 0.0,0.0 >, LIN];$
$< 0.0,0.0 >; ALL; 1; [0.0,0.2, LIN]; [0.0,0.2, LIN]]$

$rosebudbottom1 \rightarrow$
$< 4,4 > rosebudleaf1[[0.0, -5.0, LIN];$
$< 0.0,5.0 >; TOP; 1; [0.0,0.8, LIN]; [0.0,0.8, LIN]]+$
$< 1,1 > rosebudbottom2[[0.0,0.0, LIN];$
$< 0.0,0.0 >; BOTTOM; 1; [0.0,0.2, LIN]; [0.0,0.2, LIN]]$

$rosebudleaf1 \rightarrow$
$< 1,1 > rosebudleaf2[[0.0, < 70.0,90.0 >, LIN];$
$< 0.0,0.0 >; BOTTOM; 1; [0.8,0.8, LIN]; [0.8,0.8, LIN]]$

$rosebudbottom2 \rightarrow$
$< 16,16 > rosebloom1[[0.0,40.0, LIN];$
$< 5.0,80.0 >; CENTER; 1; [0.08,0.5, LIN]; [0.45,0.8, LIN]]$

Die Rosenblüte fängt hier mit dem Modul rosebudbottom1 an. Dieses erzeugt vier Kinder vom Typ rosebudleaf1, die in Länge und Breite jeweils linear wachsen, und ein Kind vom Typ rosebudbottom2. rosebudleaf1 erzeugt ein Kind vom Typ rosebudleaf2 und rosebudbottom2 erzeugt letztendlich die Blütenblätter rosebloom1.

Die Konstruktion mag kompliziert aussehen und es wird auch deswegen ein Ziel sein, den PlantAnimator derart zu verbessern, dass die Koordination von unterschiedlichen Wachstumsprozessen besser aufeinander abgestimmt werden können. Gerade wegen der Koordinierung von unterschiedlichen Wachstumsprozessen, ist nämlich die obige Konstruktion so kompliziert. Die Unterscheidung in rosebudbottom1 und rosebudbottom2 wird benötigt, damit die verdeckten Blütenblätter (rosebloom1) erst später als beispielsweise die äusseren Objekte vom Typ rosebudleaf1 generiert werden. Das breeding intervall von rosebudbottom2 startet dementsprechend später. Die Unterscheidung in rosebudleaf1 und rosebudleaf2 ist notwendig, damit die äusseren und grünen Blütenblätter sich erst gegen Ende des Wachstums-Intervalles öffnen. Die Anfangsbreite und Länge der Elemente vom Typ rosebudleaf2 entspricht daher den Endwerten von rosebudleaf1 und die Mortality von rosebudleaf1 ist auf dissappear gesetzt.

Auf jedenfall ist mit dieser Konstruktion die gewünschte Steuerung der zeitlichen Abfolge möglich, es wird aber (wie schon erwähnt) ein Ziel für die Fortführung

sein, dass auch der Benutzer die Zeitachse wesentlich intuitiver modellieren und beeinflussen kann und dass dadurch auch die Koordination der zeitlichen Abfolgen innerhalb eines Baumes verbessert werden.

Möchte man nun diese Rosenblüte als Teil einer ganzen Rose verwenden, so könnte man eine Regel wie folgt angeben:

$$
\begin{aligned}
&rosestalk4 \rightarrow \\
&\quad < 1,1 > rosebud1.pag[[0.0, 0.0, LIN]; \\
&\qquad < 0.0, 0.0 >; TOP; 1; [0.0, 0.6, LIN]; [0.0, 0.6, LIN]]+ \\
&\quad < 24, 24 > rosethorn[[< 0.0, 0.0 >, < 115.0, 115.0 >, LIN]; \\
&\qquad < 0.0, 0.0 >; CENTER; 1; [0.0, 0.2, LIN]; [0.0, 0.04, LIN]]
\end{aligned}
$$

Der oberste Rosenstengel (in diesem ersten Ansatz gab es noch keine Rekursion zur mehrfachen Hintereinandersetzung des gleichen Modules) erzeugt also an seinem oberen Ende die Rosenblüte und an beliebigen CENTER-Anknüpfpunkten 24 Dornen. Insgesamt besitzt dieses Modul über 140 Center-Anknüpfpunkte, es besteht also reichlich Möglichkeiten zur Plazierung der Dornen.

5.2 Ablauf der Animation

Das wichtigste Element für die Animation ist der sogenannte Scheduler, der als eine Art Zeitgeber dafür sorgt, dass die zu visualisierende und animierende Pflanze sich in bestimmten Abständen aktualisiert und dann gezeichnet wird. Die Animation entsteht dann durch entsprechend viele Aktualisierungen pro Sekunde (zum Beispiel 25 Aktualisierungen innerhalb einer Sekunde).

Der Scheduler schickt das Signal an die Wurzel bzw. das erste Element der Pflanze, die dieses dann an ihre jeweiligen Kinder weitergibt. Jedes Pflanzenelement prüft im Rahmen dieser Rekursion, wie sich die Parameter entwickelt haben und ob neue Kinder erzeugt werden müssen, bevor das Signal an alle Kinder weitergegeben wird.

Bei dieser Vorgehensweise und den zugrundeliegenden einfachen Berechnungen innerhalb eines Animations-Schrittes entstand dann die Idee, dass dies vielleicht Lazy Animation (siehe auch später Kapitel 8.3) ermöglicht. Das heisst, wenn es möglich ist, dass eine Pflanze trotz des Auslassens von endlich vielen Animations-Schritten effizient aktualisiert werden kann, dann kann es wahrscheinlich von Vorteil sein, ein Verfahren zu implementieren, welches in einer virtuellen Welt die Pflanzen unterscheidet, die ein Mensch sehen kann oder so sehen kann (wenn

sie zum Beispiel nicht zu weit weg ist), dass sie animiert werden müssen, und die, die nicht sichtbar sind, d. h. die nicht animiert werden müssen. Kommen die letztgenannten wieder in das Blickfeld, so können sie effizient aktualisiert werden.

5.3 Ein komplettes Beispiel

Jetzt wollen wir als komplettes Beispiel mal eine Rose betrachten, die wir ausschnittsweise schon in den vorherigen Abschnitten gesehen haben. Die Grammatik der Rose (die enthaltenen weiteren Grammatiken werden auf den nächsten Seiten erklärt) ist wie folgt definiert:

$S \rightarrow$
$< 1,1 > rosestalk0[[< -3.0,-3.0 >,< -1.0,-1.0 >, LIN];$
$< 90.0,90.0 >; ALL; 1; [0.0,0.75, LIN]; [0.0,1.0, LIN]]$

$rosestalk0 \rightarrow$
$< 1,1 > rosestalk1[[< -3.0,-3.0 >,< 7.0,7.0 >, LIN];$
$< 30.0,30.0 >; TOP; 1; [0.0,0.25, LIN]; [0.0,1.0, LIN]]$

$rosestalk1 \rightarrow$
$< 1,1 > rosestalk2[[< -3.0,-3.0 >,< 5.0,5.0 >, LIN];$
$< 135.0,135.0 >; TOP; 1; [0.0,0.25, LIN]; [0.0,1.0, LIN]]+$
$< 3,3 > rosebranch\_2.pag[[0.0,75.0, SLOWSTART1];$
$< 0.0,0.0 >; CENTER; 1; [0.0,1.0, SLOWSTART1]; [0.0,1.0, SLOWSTART1]]$

$rosestalk2 \rightarrow$
$< 1,1 > rosestalk3[[< -3.0,-3.0 >,< 7.0,7.0 >, LIN];$
$< 235.0,235.0 >; TOP; 1; [0.0,0.25, LIN]; [0.0,1.0, LIN]]+$
$< 3,3 > rosebranch\_1.pag[[0.0,65.0, LIN];$
$< 0.0,0.0 >; CENTER; 1; [0.0,1.0, LIN]; [0.0,1.0, LIN]]$

$rosestalk3 \rightarrow$
$< 1,1 > rosestalk4[[< -3.0,-3.0 >,< 10.0,10.0 >, LIN];$
$< 20.0,20.0 >; TOP; 1; [0.0,0.25, LIN]; [0.0,0.5, LIN]]+$
$< 12,12 > rosethorngreen[[< 0.0,0.0 >,< 115.0,115.0 >, LIN];$
$< 0.0,0.0 >; CENTER; 1; [0.0,0.2, LIN]; [0.0,0.04, LIN]]$

$rosestalk4 \rightarrow$
$< 1,1 > rosebud1.pag[[0.0, 0.0, LIN];$
$< 0.0, 0.0 >; TOP; 1; [0.0, 0.6, LIN]; [0.0, 0.6, LIN]]+$
$< 24,24 > rosethorn[[< 0.0, 0.0 >, < 115.0, 115.0 >, LIN];$
$< 0.0, 0.0 >; CENTER; 1; [0.0, 0.2, LIN]; [0.0, 0.04, LIN]]$

$rosethorn \rightarrow NULL$

$rosethorngreen \rightarrow NULL$

Eine Rose besteht hier also aus dem Stengel, Seitenzweigen, Dornen und der Blüte. Dies ist gleich ein Beispiel für die im ersten Ansatz noch fehlende Rekursion im PlantAnimator, die dafür sorgt, dass für den Stengel fünf Module benutzt werden müssen. Andererseits hat man hier dafür die Freiheit, an den verschiedenen Modulen des Stengels verschiedene Kinder zu erzeugen. In der Mitte zum Beispiel zwei unterschiedliche Arten von Seitenzweigen und oben zwei unterschiedliche Sorten von Dornen. Mit den Seitenzweigen werden wir uns gleich beschäftigen, die Dornen unterscheiden sich nur in der benutzten Texture. Das Funktions-Alias $SLOWSTART1$ steht hier für eine benutze Sigmoid-Funktion (siehe auch Bild 4.9 auf Seite 46).

rosebranch_1.pag:

$S \rightarrow$
$< 1,1 > rosebranch1[[< 1.0, 1.0 >, < 0.0, 0.0 >, LIN];$
$< 0.0, 0.0 >; ALL; 1; [0.0, 0.065, LIN]; [0.0, 0.4, LIN]]$

$rosebranch1 \rightarrow$
$< 1,1 > roseauxleaf1[[< 15.0, 15.0 >, < 15.0, 15.0 >, LIN];$
$< 90.0, 90.0 >; BOTTOM; 1; [0.0, 0.07, LIN]; [0.0, 0.11, LIN]]+$
$< 1,1 > roseauxleaf1[[< 15.0, 15.0 >, < 15.0, 15.0 >, LIN]; < 270.0, 270.0 >;$
$BOTTOM; 1; [0.0, 0.07, LIN]; [0.0, 0.11, LIN]]+$
$< 1,2 > roseleaf1[[< 1.0, 10.0 >, < -20.0, -10.0 >, LIN];$
$< -20.0, 30.0 >; TOP; 1; [0.0, 0.65, SLOWSTART1]; [0.0, 0.65, SLOWSTART1]]+$
$< 2,2 > rosestalk5[[< 14.0, 17.0 >, < 65.0, 70.0 >, LIN]; < 85.0, 95.0 >;$
$CENTER; 1; [0.0, 0.05, SLOWSTART1]; [0.0, 0.05, SLOWSTART1]]$

$rosestalk5 \rightarrow$
$< 1, 1 > roseleaf1[[< 0.0, 0.0 >, < 0.0, -0.0 >, LIN];$
$< 75.0, 75.0 >; BOTTOM; 1; [0.0, 0.7, SLOWSTART1]; [0.0, 0.7, SLOWSTART1]]$

$roseauxleaf1 \rightarrow NULL$

$roseleaf1 \rightarrow NULL$

Ein wie oben beschriebener Seitenzweig sieht wie in Bild 5.7 aus, an seinem Anfang werden zwei kleine Hilfsblätter erzeugt, an seinem Ende ein oder zwei Blätter und in der Mitte zwei kleinere Zweige, die jeweils wieder ein einzelnes Blatt enthalten.

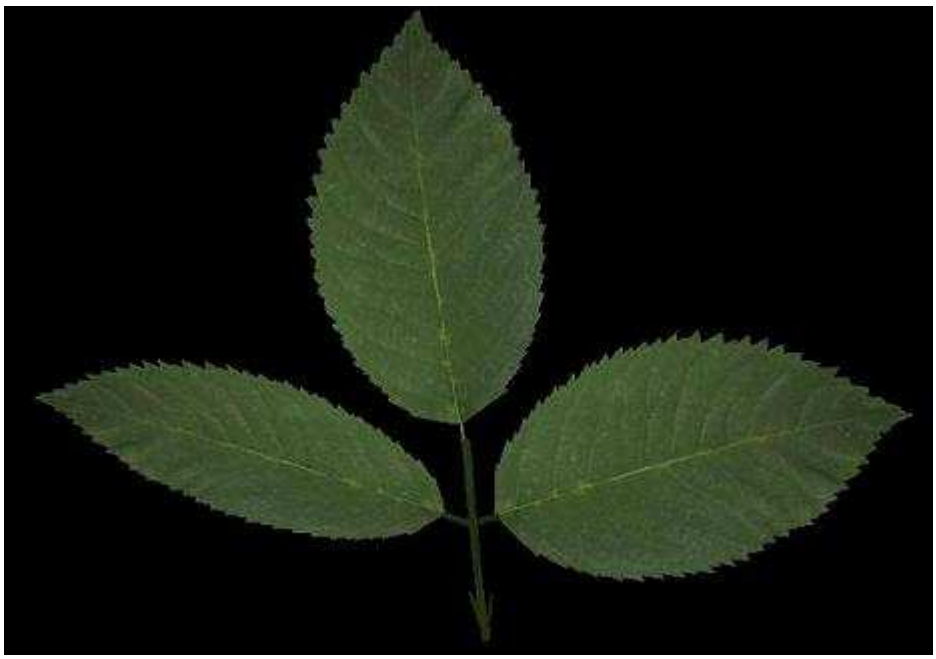


Abbildung 5.7: Ein Rosenzweig

rosebranch_2.pag:

$S \rightarrow$
$< 1,1 > rosebranch2[[< 1.0, 1.0 >, < 0.0, 0.0 >, LIN];$
$< 0.0, 0.0 >; ALL; 1; [0.0, 0.07, LIN]; [0.0, 0.8, LIN]]$

$rosebranch2 \rightarrow$
$< 1,1 > roseauxleaf1[[< 15.0, 15.0 >, < 15.0, 15.0 >, LIN];$
$< 90.0, 90.0 >; BOTTOM; 1; [0.0, 0.07, LIN]; [0.0, 0.11, LIN]]+$
$< 1,1 > roseauxleaf1[[< 15.0, 15.0 >, < 15.0, 15.0 >, LIN];$
$< 270.0, 270.0 >; BOTTOM; 1; [0.0, 0.07, LIN]; [0.0, 0.11, LIN]]+$
$< 1,1 > roseleaf1[[< 3.0, 3.0 >, < -10.0, -10.0 >, LIN];$
$< 0.0, 0.0 >; TOP; 1; [0.0, 0.7, SLOWSTART1]; [0.0, 0.7, SLOWSTART1]]+$
$< 4,4 > rosestalk5[[< 14.0, 17.0 >, < 65.0, 70.0 >, LIN];$
$< 85.0, 95.0 >; CENTER; 1; [0.0, 0.05, SLOWSTART1]; [0.0, 0.05, SLOWSTART1]]$

$rosestalk5 \rightarrow$
$< 1,1 > roseleaf1[[< 0.0, 0.0 >, < 0.0, -0.0 >, LIN];$
$< 75.0, 75.0 >; BOTTOM; 1; [0.0, 0.7, SLOWSTART1]; [0.0, 0.7, SLOWSTART1]]$

$roseauxleaf1 \rightarrow NULL$

$roseleaf1 \rightarrow NULL$

Ein wie hier beschriebener Seitenzweig sieht wie in Bild 5.8 aus. Dieser Zweig wird insgesamt länger und hat von der Form her ein etwas anderes Aussehen. Wie schon bei dem ersten Zweig werden hier auch zwei Hilfsblätter an dem unteren Ende des Zweiges erzeugt, dann aber nur genau ein Blatt am Ende des Zweiges und vier Seitenverzweigungen, die jeweils ein Blatt enthalten.

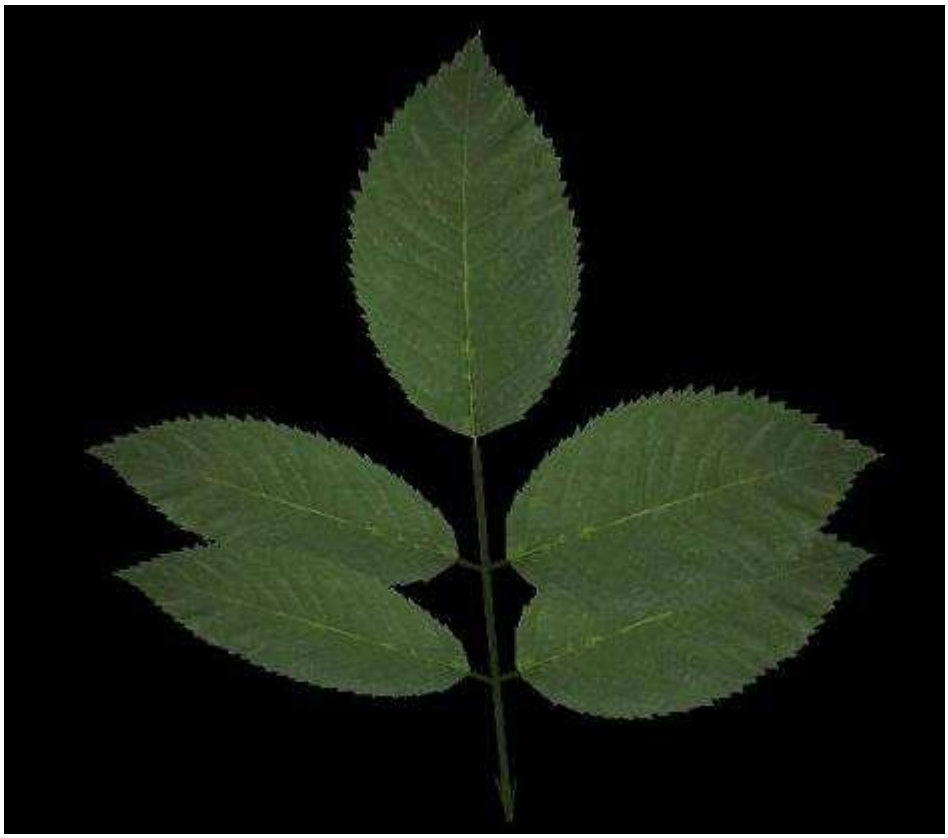


Abbildung 5.8: Ein anderer Rosenzweig

rosebud1.pag:

$S \rightarrow$
$< 1,1 > rosebudbottom1[[< 0.0, 0.0 >, < 0.0, 0.0 >, LIN];$
$< 0.0, 0.0 >; ALL; 1; [0.0, 0.2, LIN]; [0.0, 0.2, LIN]]$

$rosebudbottom1 \rightarrow$
$< 4,4 > rosebudleaf1[[0.0, -5.0, LIN];$
$< 0.0, 5.0 >; TOP; 1; [0.0, 0.8, LIN]; [0.0, 0.8, LIN]]+$
$< 1,1 > rosebudbottom2[[0.0, 0.0, LIN];$
$< 0.0, 0.0 >; BOTTOM; 1; [0.0, 0.2, LIN]; [0.0, 0.2, LIN]]$

$rosebudleaf1 \rightarrow$
$< 1,1 > rosebudleaf2[[0.0, < 70.0, 90.0 >, LIN];$
$< 0.0, 0.0 >; BOTTOM; 1; [0.8, 0.8, LIN]; [0.8, 0.8, LIN]]$

$rosebudbottom2 \rightarrow$
$< 16, 16 > rosebloom1[[0.0, 40.0, LIN];$
$< 5.0, 80.0 >; CENTER; 1; [0.08, 0.5, LIN]; [0.45, 0.8, LIN]]$

Abbildung 5.9: Die Rosenblüte

Wie genau das Aussehen dieser Rosenblüte zustande kommt und wieso die Definition etwas komplexer aussieht, wurde auf Seite 72 schon erklärt, das Gesamtergebnis kann in Bild 5.9 betrachtet werden.

Das Wachstum dieser Rose kann dann im Plantanimator betrachtet werden, wir haben allerdings auch eine Sequenz von 630 Einzel-Bildern erzeugt, die das Wachstums ebenfalls demonstrieren und von denen wir uns einige jetzt anschauen wollen:

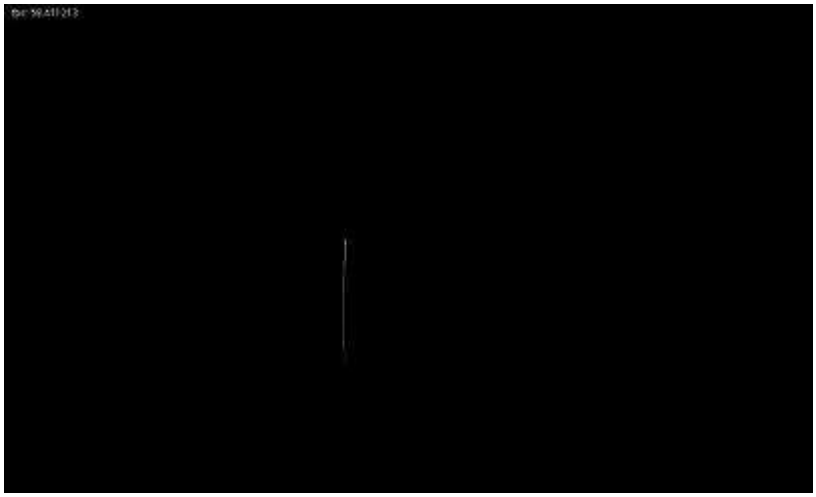


Abbildung 5.10: Die Rose nach 80 Zeitschritten

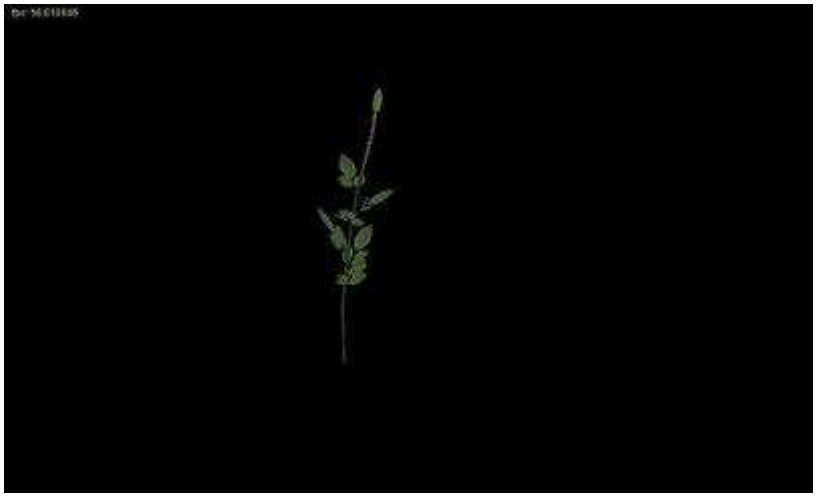


Abbildung 5.11: Die Rose nach 160 Zeitschritten

Man sieht zwischen den Schritten 80 und 160 ein ordentliches Wachstum der gesamten Pflanze, die Blüte ist noch geschlossen.

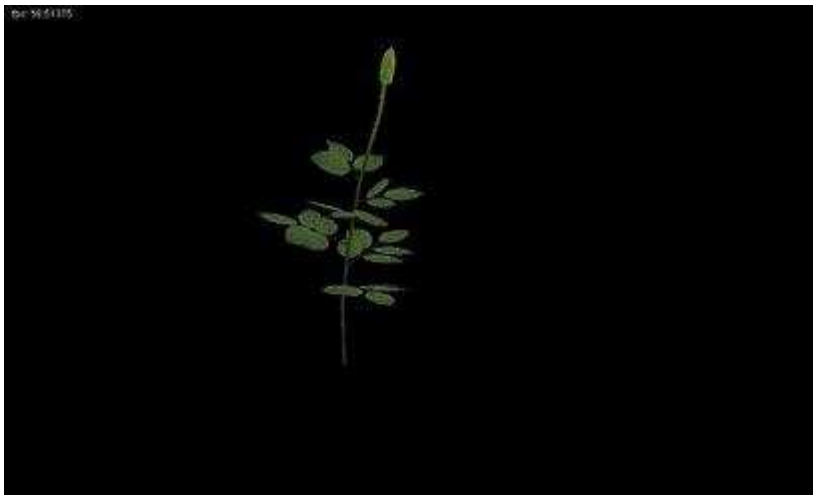


Abbildung 5.12: Die Rose nach 240 Zeitschritten

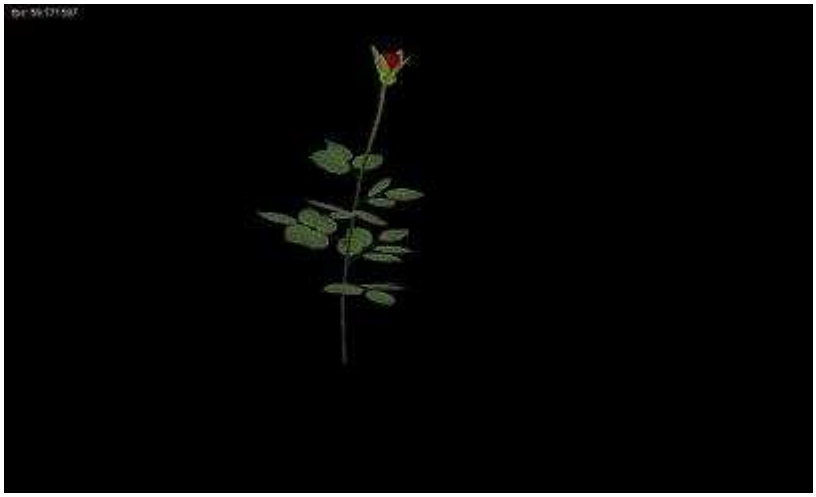


Abbildung 5.13: Die Rose nach 320 Zeitschritten

Der Unterschied zwischen Schritt 160 und 240 ist hauptsächlich noch Wachstum in den Seitentrieben und Winkeländerungen (die Seitentriebe klappen etwas von der Pflanze ab). In Schritt 320 beginnen dann die äusseren Blütenblätter sich zu öffnen.

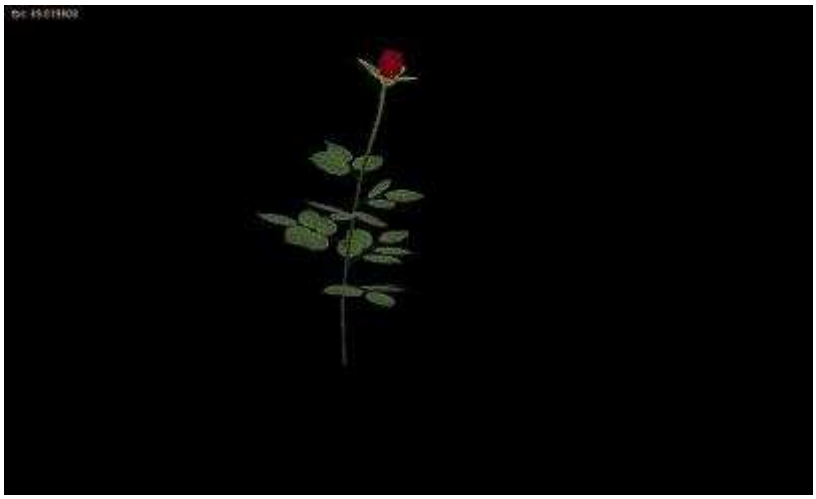


Abbildung 5.14: Die Rose nach 400 Zeitschritten

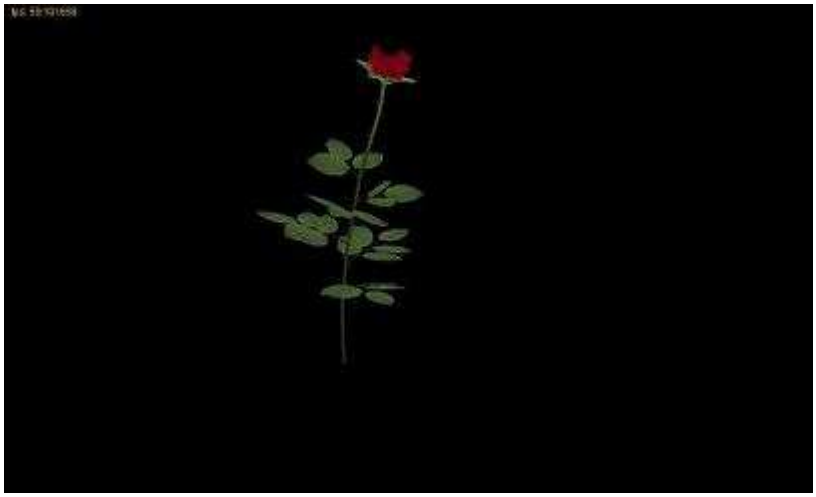


Abbildung 5.15: Die Rose nach 630 Zeitschritten

Nach 400 Zeitschritten haben sich die äusseren Blütenblätter geöffnet, nun beginnen gleich die inneren Blütenblätter sich noch etwas zu öffnen. Nach dem letzten Schritt haben sich auch die inneren Blütenblätter sehr langsam im Winkel von 40 Grad geöffnet.

5.4 Vorteile

Während das in diesem Kapitel beschriebene Modellieren der Pflanze sich dem Bereich der regelbasierten Objekterzeugung zuordnet, werden hier beim Modellieren gleich noch für die Animation von Pflanzenwachstum wichtige Aspekte mit bearbeitet. Dies erlaubt uns, wie an dem vorherigen Beispiel gezeigt, mit recht einfachen Mitteln die Wachstumsanimation einer Pflanze anzufertigen.

Im Gegensatz zu beispielsweise differentiellen L-Systemen haben wir auch eine grössere Flexibilität mit den Ansatzpunkten und auch den Zeitpunkten, wann ein Seitentrieb erzeugt werden kann. Das Erzeugen von Seitentrieben geschieht bei uns nicht nur, wenn wie bei den differentiellen L-Systemen einer der Wachstums-Parameter aus seiner Domain herausläuft. Dies dürfte für den Benutzer nur schwer zu kontrollieren sein und es ist sicher auch nicht effizient, wenn bei jedem Erzeugen eines Seitentriebes eine Ersetzung im L-System stattfindet. Es müssen bei uns auch keine Differentialgleichungen ausgewertet oder durch den Benutzer angegeben werden.

Am Schluss von [PHM93] (die Veröffentlichung zu den differentiellen L-Systemen) stehen ausserdem noch vier Punkte zu möglichen Verbesserungen, die wir überwiegend erfüllt haben.

1. Kombinierte differential-algebraische Spezifikation von kontinuierlichen Aspekten
2. Einbeziehen von stochastischen Aspekten
3. Entwicklung einer Simulations-Software
4. Gesteigerte Realitätsnähe der Modelle

Zu Punkt 1 und der Frage, was die differential-algebraische Spezifikation ist, sagt beispielsweise Wikipedia: In einer differential algebraischen Gleichung (auch Algebro-Differentialgleichung oder Deskriptor-System) sind gewöhnliche Differentialgleichungen und algebraische (d. h. hier: ableitungsfreie) Nebenbedingungen gekoppelt und werden als eine Gleichung bzw. Gleichungssystem aufgefasst. Der PlantAnimator geht sogar noch einen Schritt weiter, indem kontinuierlichen Aspekte gar nicht erst durch Differentialgleichungen beschrieben werden. Die Punkt 2 und 3 werden sicher auch erfüllt, wobei eigentlich der Hauptpunkt von Prusinkiewicz et al zu Punkt 3 die Integration eines Differentialgleichungs-Lösers in deren System, welches auf parametrischen L-Systemen aufbaut, gewesen ist. Punkt 4 beinhaltet praktische Probleme wie das verbesserte Modellieren und das Verwelken, was im PlantAnimator auch behandelt wird.

Insgesamt haben wir ein System, welches vom reinen Modellieren der Pflanze sicher dem aktuellen Stand der Technik bzw. Wissenschaft entspricht und welches mit wenigen zusätzlichen und einfachen Parametern und Mechanismen, die Animation einer wachsenden Pflanze erzeugen kann. Allerdings haben wir an dem Beispiel auch schon gesehen, dass für manche Sachen wie zum Beispiel auch die zeitliche Abstimmungen von den Wachstumsprozessen innerhalb der Pflanze, doch etwas umständlich gearbeitet werden musste und es zeichnet sich ab, dass auch nicht alles, was für dieses erste System vorgesehen war, auch wirklich so einfach lösbar ist, wie wir uns es zu Beginn vorgestellt haben. Dies bringt uns zu dem nächsten Abschitt und später auch zu der Weiterentwicklung des PlantAnimator.

5.5 Verbesserungsfähig

Im Folgenden werden wir uns mit einigen Dingen beschäftigen, die an dem PlantAnimator und den dahinter liegenden Ideen noch verbessert werden können. Dabei werden wir den Begriff der umwelt-getriebene Animation nicht noch einmal gesondert erwähnen, da diese bisher noch gar nicht berücksichtigt wurde und für sich genommen, eine grössere Erweiterung des Tools darstellt.

5.5.1 Durchdringung von Pflanzenelementen

Das Erkennen und Behandeln von gegenseitigen Durchdringung der verschiedenen Pflanzen-Elementen stellt eine Herausforderung an die Computer-Technik dar. Bevor man sich allerdings mit der Lösung befasst, sollte man ein paar grundsätzliche Fragen klären:

1. Ist es für die Realitätsnähe oder die vornehmlich gewünschten Einsatzgebiete der Pflanzen notwendig, dass Durchdringungen verhindert werden?
2. Wenn man die Durchdringungen verhindern will, muss dieses schon bei dem Modellieren der Pflanze geschehen oder kann eigentlich nur das Visualisierungs-System dafür sorgen?

Über die erste Frage kann man sich sicherlich trefflich streiten, die zweite Frage lässt sich sicherlich einfach beantworten, wenn man bedenkt, dass nach dem Modellieren noch weitere Einflüsse dazukommen (wie zum Beispiel Tropismen oder umwelt-getriebene Animation). Daher macht es nur wenig Sinn, bei dem Modellieren der Pflanze und der Wachstumsanimation sich über die Durchdringungen schon Gedanken zu machen. In dem PlantAnimator wird dieser Aspekt noch

gar nicht berücksichtigt und wird von uns auch nicht als zwingend notwendig betrachtet.

Wollte man es einbauen, so würden sich gleich weitere verschiedene Fragen stellen, zum Beispiel:

1. Wird die möglichen Durchdringungen schon im Scheduler (beim Aktualisieren der Pflanze) oder erst im Visualizer behandelt?

2. Muss der Ablauf des entsprechenden Prozesses dann vielleicht eher einer Breiten-Wanderung im gedachten Graphen der Pflanzenelemente entsprechen? Um dadurch eventuelle Beeinflussungen von nebeneinanderliegenden Pflanzenelementen gleich berücksichtigen zu können. Momentan entspricht der Prozess einer Tiefen-Wanderung, welches sich ohne eine zusätzliche Speicherstruktur und nur mit Rekursion umsetzen lässt.

3. Wie behandelt man das Aufreten einer Durchdringung genau? Wird nur ein Kandidat verbogen, werden beide Kandidaten verbogen und welche Faktoren werden berücksichtigt, wenn es darum geht, wer und wie weit verbogen wird?

4. Wie behandelt man einen möglichen Teil-Zyklus, das heisst wenn Seitentriebe mit einem nebenliegenden und schon behandelten Teil der Pflanze in Konflikt bezüglich einer möglichen Durchdringung kommen?

5. Ist es effizient machbar, dass bis zu mehrere Millionen Blätter auf Durchdringungen getestet werden?

Die ersten beiden dieser Fragen lassen sich sicherlich einfach klären und werden den Anwender auch nicht interessieren, die dritte Frage ist von praktischer Natur und die Klärung dieser Frage wird das realitätsnahe Modellieren und Verhalten einer Pflanze mitbestimmen. Die zu berücksichtigenden Faktoren beinhalten auf der einen Seite geometrische Informationen wie zum Beispiel die Breite oder noch besser den Durchmesser des Planzen-Elementes, aber auf der anderen Seiten sicher auch bisher noch gar nicht modellierte Faktoren, die zu dem entsprechendem Material der Pflanzen-Elemente gehört.

Die vierte Frage ist wiederum für Anwender nicht sonderlich interessant aber in den vielen Fällen sicher auch nicht einfach lösbar. In dieser Situation können verschiedene Fälle auftreten, die sich jeweils in der Ordnung oder Tiefe in der Modulhierarchie der beiden beteiligten Module unterscheiden. Besondere Aufmerksamkeit müsste man dem Fall widmen, dass die Ordnung des gerade betrachteten Modules niedriger ist als die Ordnung des schon aktualisierten Modules. Dies würde bedeuten, dass das gerade betrachtete Modul einen grösseren Durchmesser

hat als das schon vorher aktualisierte Modul. In diesem Falle müsste das schon aktualisierte Modul neu berechnet und als Folge der Kollision neu positioniert werden. Selbst, wenn als Resultat aus der zweiten Frage eine Breiten-Wanderung benutzt wird, bedeutet dies nicht, dass solche Fälle gar nicht mehr auftreten können.

Die fünfte Frage führt uns in den Bereich der Kollisionserkennung und bringt uns zu Fragestellungen, wie man die Kollisions-Erkennung effizient für möglicherweise sehr viele Objekte lösen kann. Auf diesem Gebiet gibt es schon einige gute Ansätze, doch es bleibt zu erwarten, dass die Kosten (selbst bei einem einfachen Modell, was vielleicht nur mit achs-orientierten Bounding-Boxen arbeitet) den Nutzen nicht tragen werden.

5.5.2 Bessere Kontrolle der Zeitachse

An dem Beispiel der Rose haben wir gesehen, dass man mit der Zeitachse für die Animation durchaus kreativ arbeiten kann. Dort wurde eine Unterscheidung in rosebudbottom1 und rosebudbottom2 gemacht, damit die verdeckten Blütenblätter vom Typ rosebloom1 erst später als beispielsweise die äusseren Objekte vom Typ rosebudleaf1 generiert werden. Zu diesem Zwecke startete das breeding intervall von rosebudbottom2 erst später. Eine weitere Unterscheidung in rosebudleaf1 und rosebudleaf2 wurde gemacht, damit die die äusseren und grünen Blütenblätter sich erst gegen Ende des Wachstums-Intervalles öffnen. Die Anfangsbreite und Länge der Elemente vom Typ rosebudleaf2 entspricht daher den Endwerten von rosebudleaf1 und die Mortality von rosebudleaf1 ist auf dissappear gesetzt.

Ähnliche Effekte hätte man sicher auch durch angepasste und sehr spezielle Funktionen für die Wachstumsparameter erzielen können, aber auch dann wäre die Steuerung der Zeitachse und insbesondere die Abstimmung der einzelnen Wachstums-Prozesse innerhalb der Pflanze aufeinander, sicher nicht einfach und intuitiv gewesen. Es sollte daher ein Ziel für die Weiterentwicklung sein, dass die Zeitachse noch mehr betont wird und dass sie noch besser vom Anwender modelliert und beeinflusst werden kann.

5.5.3 Verjüngung der Pflanzen-Objekte

Bei dem Entwurf des PlantAnimator sollte die Steuerung der Abnahme der Breite über die Länge eines Modules hinweg durch einen weiteren Faktor in der Grammatik umgesetzt werden. Damit sollte die Wiederverwendbarkeit eines Modules verbessert werden. Im Rahmen der praktischen Umsetzung wurde dieser Faktor

dann aber ausgeklammert und nur die Werte in der Grammatik für die Breite benutzt. Das Ausklammern der Verjüngung lag an der schwierigen Umsetzung dieses Konzeptes im Rahmen des PlantAnimators und damit unter Benutzung von affinen Transformation. Andererseits gibt es auch Fälle, in denen es nicht sinnvoll ist, nur mit festen Werten für die Breite in der Grammatik zu arbeiten.

Es ist nämlich ein Problem, wenn man bei Fortsetzungen eines Pflanzenelementes an einem Anknüpfpunkt der Kategorie TOP Start- und End-Wert für die Breite vorgibt. Stattdessen sollte die Startbreite eines Elementes von der Endbreite des Vorgänger-Elementes abhängen. Die Endbreite sollte dann entweder aus der Geometrie des Modules abgeleitet oder durch einen Faktor, der für eine ungleichmässige Skalierung verwendet werden kann, angegeben werden.

Genauer werden wir uns mit diesem Thema noch in dem Kapitel 7 befassen.

5.5.4 Wiederverwendbarkeit der Module

Ein Vorteil, den die regelbasierte Objekterzeugung bieten soll, ist die erhöhte Wiederverwendbarkeit von Modulen und Definitionen. Dies wurde im PlantAnimator zwar schon aufgegriffen, aber noch nicht zufriedenstellend gelöst. Für die Wiederverwendbarkeit von Modulen ist es nämlich besonders interessant, wenn möglichst viele Parameter in der Grammatik stehen und damit den Modulen dynamisch zugeordnet werden.

Folgende Werte sind momentan über die Grammatik beeinflussbar:

- Die Länge des Modules
- Die Breite des Modules
- Der Winkel zwischen zwei Pflanzen-Modulen
- Der Drehwinkel zwischen zwei Pflanzen-Modulen
- Die Anzahl und Platzierung der Kinder-Module an Ansatzpunkten

Wobei die grundsätzliche Länge und Breite von Modulen natürlich durch deren Geometrie angegeben wird, aber durch die Angabe der Skalierungsfaktoren in der Grammatik kann darauf aufbauend die tatsächliche Länge und Breite bestimmt werden. Hier offenbart sich auch ein Vorteil der Tatsache, dass alle Module auf die Länge 1 skaliert werden. Dadurch kann der Benutzer nämlich die Längenverhältnisse innerhalb der Pflanze aufeinander abstimmen. Wird einem Blatt die

Länge 0,1 und einem Zweig die Länge 1,0 zugewiesen, dann ist der Zweig zehnmal so lang wie das Blatt. Die Längenwerte innerhalb der Pflanze werden also miteinander vergleichbar.

Folgende Werte sind momentan den Modulen zugeordnet:

- Zeiträume für Wachsen und Bilden von Kinder-Modulen
- Verhalten beim Ende des Wachstumes
- Farb- und Texture-Informationen

Das die Verankerung der Zeiten in den Modulen nicht die beste Lösung ist, haben wir in 5.5.2 schon gesehen, aber auch bei den anderen Werten kann man sich sicher gut vorstellen, dass diese durch das Einarbeiten in die Grammatik und das Loslösen von den Modulen noch besser beeinflussbar wären und damit auch die Wiederverwendbarkeit der Module steigt.

5.5.5 Schöneres Aussehen

Ein schöneres Aussehen ist im Allgemeinen eine Frage der Zeit, die man in das Modellieren steckt oder stecken kann. Auch bei den Verfahren der regelbasierten Objekterzeugung gilt, dass man mit mehr Modellier-Zeit ein schöneres Ergebnis erzielen kann.

Kapitel 6

Umwelt-getriebene Animation

Unter umwelt-getriebener Animation verstehen wir hier Animation, die nicht aus einer Pflanze selbst herauskommt und die auch nicht zu Änderungen in dem Wachstum der Pflanze führt. Das Gegenteil davon wäre dann alles, was das Wachstum oder besser die Form einer Pflanze beeinflusst. Ein Musterbeispiel für die umwelt-getriebene Animation wäre der Wind, der eine Pflanze hin- und her wiegt. Allerdings muss unterschieden und beachtet werden, dass ein längerfristig gleichbleibender Wind auch die Gesamtform einer Pflanze beeinflussen kann (siehe auch in Kapitel 2.2). Ausserdem kann durch das mögliche Abbrechen von Ästen durch Wind oder Stürme eine gewisse Unregelmässigkeit in Pflanzen eingeführt werden. Bisher ist die umwelt-getriebene Animation das Gebiet in der Animation von Pflanzen, was am wenigsten untersucht wurde.

In diesem Kapitel werden wir zuerst mit den umwelt-getriebenen Animations-Möglichkeiten für ältere Ansätze zum Modellieren von Pflanzen beschäftigen. Danach wollen wir einmal die Animations-Möglichkeiten in dem in der Praxis sehr verbreiteten Programm Maya betrachten. Am Ende des Kapitels wollen wir uns dann Gedanken darüber machen, wie die umwelt-getriebene Animation in den PlantAnimator integriert und umgesetzt werden kann.

Die umwelt-getriebene Animation wird in diesem Kapitel hauptsächlich anhand des Standard-Beispiel **Wind** beschrieben, weitere Anwendungsmöglichkeiten werden wir in Abschnitt 9.2 noch beleuchten.

6.1 Umwelt-getriebene Animation bei prozeduralen Methoden

Eigentlich kann die umwelt-getriebene Animation nur dann funktionieren, wenn durch die benutzte Erzeugungs-Methode der Pflanzen auch eine echte Geometrie der Komponenten beschrieben wird.

Sicherlich wird man sofort einsehen, dass die Ablenkung, die ein Wind bei einer Pflanze hervorruft, ganz wesentlich auch von dem Material und der Dicke des abzulenkenden Objektes bzw. der Komponente abhängt. Daher eignet sich nicht jede prozedurale Methode, um die Pflanze auch umwelt-getrieben animieren zu können. Besonders schwierig sollte dies bei der Umsetzung mit zellulären Automaten oder Partikelsystemen werden. Diese beiden Ansätze bieten nur wenige Möglichkeiten, den Zusammenhang von verschiedenen Komponenten zu beschreiben. Damit wird es nur schwer möglich sein, wenn man den Einfluss von Wind auf für Pflanzen typische Komponenten wie Äste und Blätter beschreiben will.

Für fraktale Modelle gibt es zwar die sogenannte fraktionale brownsche Bewegung (wie in [NTT92] beschrieben), die auf der aus vielen Bereichen bekannten brownschen Bewegung aufbaut. Diese ist aber wahrscheinlich für die einfache Modellierung der umwelt-getriebenen Animation ungeeignet.

Die Brownsche Bewegung beschreibt das zufällige Bewegen von Partikeln, die sich in einer Flüssigkeit oder einem Gas befinden, bzw. dessen mathematisches Modell. Das mathematische Modell hat einige weitere Anwendungsmöglichkeiten in der realen Welt gefunden wie zum Beispiel die Veränderungen von Börsenkursen.

Während die brownsche Bewegung eine zufällige Bewegung modelliert, ist die fraktionale brownsche Bewegung nicht zufällig, sondern ein späterer Schritt kann von einem oder mehreren früheren Schritten abhängen (was für die umwelt-getriebene Animation zum Beispiel durch Wind kein Nachteil sein muss). Insgesamt steht hinter dem ganzen Konzept der brownschen Bewegungen aber wieder ein sehr mathematisches Konzept (siehe auch [Wik08a]). Dies eignet sich daher sicherlich weniger für eine intuitive Modellierung und ist auch nicht auf die Interaktion mit der Umwelt und sich dort ändernden Parametern ausgelegt.

Insgesamt muss man auch darauf hinweisen, dass die prozeduralen Methoden sich in der Praxis nicht durchgesetzt haben. Daher gibt es auch praktisch keine Forschung in dem Gebiet, wie die mit prozeduralen Methoden generierten Pflanzen umwelt-getrieben animiert werden können.

6.2 Animation basierend auf der Interaktion von L-Systemen und Vektor-Feldern

In [NTT92] wird die Möglichkeit untersucht, wie man eine mit einem L-System beschriebene Pflanze mit äusseren Kräften wie Wind verbinden kann.

Die hier beschriebenen L-Systeme bauen auf 'timed parametric context free l systems' auf. Timed L-Systeme sind eine Art Vorgänger von den differentiellen L-Systemen, sie besitzen eine zu jedem Symbol gehörige Zeitangabe. Eine Produtionwird nur ausgeführt, wenn das Alter eines Symboles das in der Produktion angegebene Alter erreicht oder überschreitet. Ein einfaches Beispiel dazu sehen wir in Bild 6.1. Dort ist c ein Symbol, welches ein geometrisches Objekt erzeugt, [und] sind die bekannten Stack-Operationen eines L-Systemes (zum Speichern und Laden von Positionen) und + rotiert den Turtle um seine Achse.

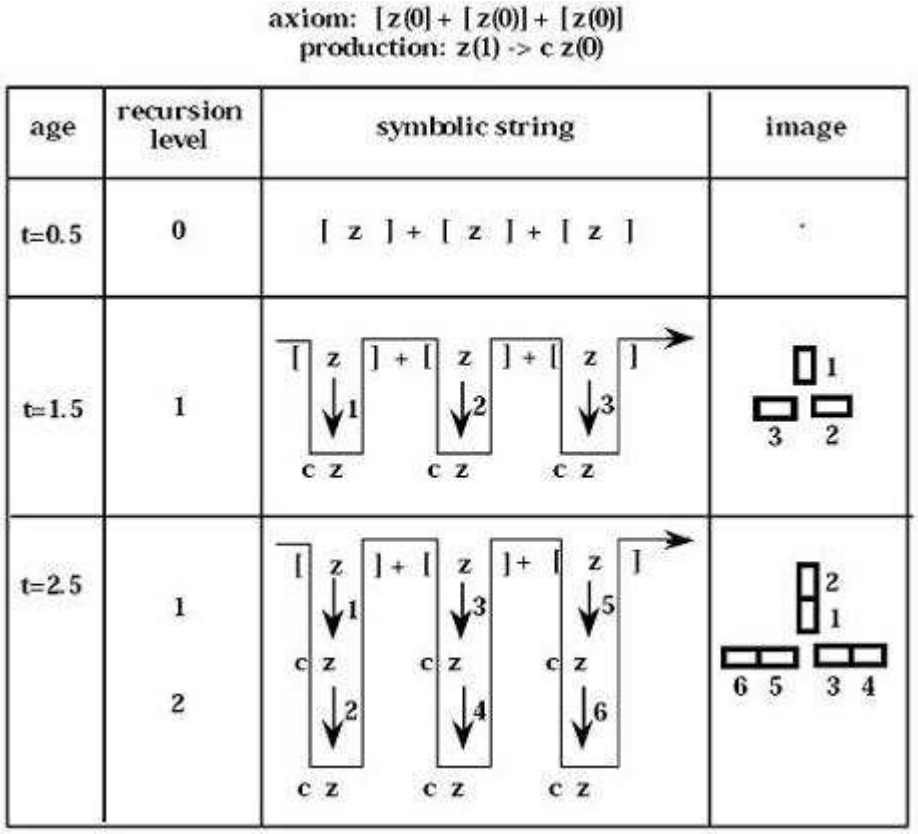


Abbildung 6.1: Aussehen und Vorgehensweise von Timed L-Systemen (aus [NTT92])

Hier wird dieses Vorgehen erweitert zu folgenden L-Systemen:

1.	V	Das Alphabet
2.	C	Eine Menge von Konstanten
3.	R^+	Die Menge der positiven Real-Zahlen
4.	R^3	Der 3D-Raum
5.	Σ	Die Menge der formalen Parameter
6.	$E(\Sigma)$	Arithmetische Ausdrücke
7.	$C(\Sigma)$	Logische Ausdrücke
8.	ω	Das Start-Axiom
9.	P	$P = \{p_{a,i} \mid a \in V, i \in N\}$ die Menge der Produktionen
10.	Π	$p_{a,i} \rightarrow [0,1]$ eine pseudo-statistische Verteilung mit $\Sigma\pi(p_{a,i}) = 1$
11.	$(a, x_0, ...x_6)$	$\in V \times R^7$ parametrisierte Symbole

Der erste Parameter x_0 in Zeile 11 steht für die Zeit oder das Alter des Symboles, während die Interpretation der anderen Parameter nicht festgelegt ist und von der Semantik abhängen, die ihnen gegeben wird.

Sei $P \in V \times R^7 \times C(\Sigma) \times \Pi \times (V \times E(\Sigma)^7)^*$ und $p_{a,i} \in P$, dann gilt:

$$p_{a,i} : (a, \alpha, x_1, ..., x_6) \quad \begin{matrix} Cond(\alpha, x_1, x_2, x_3, T) \\ - - - - - - - - - - - - > \\ \Pi(p_{a,i}) \end{matrix} \quad \begin{matrix} (a_1, f_{10}, ..., f_{16}) \\ . \\ . \\ . \\ a_n, f_{n0}, ..., f_{n6}) \end{matrix}$$

Hierbei ist

α	das maximale Symbol Alter
f_{j0}	das Alter des Symboles a_j zu Beginn
x_1, x_2, x_3	3 Parameter
$f_{j0} + t$	Lokales Alter des Symboles
T	Globale Zeit
$f_{j1}...f_{j3}$	Parameter-Ausdrücke
$f_{j4}...f_{j6}$	Wachstums-Funktionen
f_{jk}	$f_{jk}(f_{j0}, x_1, x_2, x_3, t, T)$ eine bedingte und pseudo-statistische Produktion

Für die Ableitungsfunktion $D : (V \times R^7)^* \times R \rightarrow (V \times R^7)^*$ gilt:

- Die Entwicklung der Symbole ist unabhängig voneinander,
  $D((a_1, x_0, ...x_6)...(a_n, x_0, ..., x_6), t) = D((a_1, x_0, ..., x_6), t)...D((a_n, x_0, ..., x_6), t)$
- Für das Wachstum eines Symboles vor dem End-Alter gilt: If $x_0 + t < \alpha$ Then $D((a_1, x_0, ..., x_6), t) = (a, x_0 + t, x_1, x_2, x_3, f_4, f_5, f_6)$ mit $f_j = f_j(x_0 + t, x_1, x_2, x_3, T), j = 4, 5, 6$ als Wachstumsfunktionen.
- Das Anwenden der stochastischen Produktion am End-Alter: If $x_0 + t > \alpha$ und $Cond(\alpha, x_1, x_2, x_3, T) =$ TRUE mit der Wahrscheinlichkeit $\Pi(p_{a,i})$ Then $D((a, x_0, ..., x_6), t) = D((a_1, f_{10}, ..., f_{13}, x_4, x_5, x_6)...(a_n, f_{n0}, ..., f_{n3}, x_4, x_5, x_6), t - (\alpha - x_0))$

Wie genau die Wachstums-Funktionen oder die ganzen Parameter der Symbole aussehen oder auch nur aussehen könnten, wird in dem Papier leider nicht angegeben. Man sieht lediglich, dass die Wachstumsfunktionen während des normalen Wachstumes (also solange $x_0 + t < \alpha$ gilt) fortwährend auf die beiden Zeit-Parameter (Alter des Symboles $x_0 + t$ und globale Zeit T) und die ersten drei Parameter x_1, x_2, x_3 (die sich nicht verändern) angewendet werden: $f_j = f_j(x_0 + t, x_1, x_2, x_3, T), j = 4, 5, 6$

Die äusseren Kräfte stellt man sich hier dann als ein zeit- und ortsabhängiges Vektorfeld von Kräften vor. Die Bewegungen werden dann durch ein Gleichungssystem von Differentialgleichungen wie folgt beschrieben:

$\ddot{x} = f_x(x, y, z, \dot{x}, \dot{y}, \dot{z}, t, X, Y, Z)/m$
$\ddot{y} = f_y(x, y, z, \dot{x}, \dot{y}, \dot{z}, t, X, Y, Z)/m$
$\ddot{z} = f_z(x, y, z, \dot{x}, \dot{y}, \dot{z}, t, X, Y, Z)/m$

$x, y, z : Position$
$\dot{x}, \dot{y}, \dot{z} : Geschwindigkeit$
$X, Y, Z : Anfangsposition$
$m : Masse$

Zum Lösen der Differentialgleichungen wird das RungeKutta-Verfahren der Ordnung 4 (siehe auch Seite 47) benutzt. RungeKutta berechnet aufbauend auf einem alten Wert zum Zeitpunkt t_0 einen neuen Wert $t_0 + dt$, indem mehrere Zwischen-Steigungen berechnet werden und diese dann gewichtet zu einer Gesamt-Steigung addiert werden.

In einem Beispiel, welches in [NTT92] angegeben wird, wird ein sich bewegender Wirbelwind in Vebindung mit einem periodischen Wind durch die folgenden Gleichungen angegeben:

$$f_x = -z/((x + 50 - 1,8 * T)^2 + z^2 + 0,5) + a + b + sin(c * T)$$
$$f_z = (x + 50 - 1,8 * T)/((x + 50 + 1,8 * T)^2 + z^2 + 0,5) + d + e + sin(f * T)$$

Über die Parameter a, b, c, d, e und f wird leider nichts weiter ausgesagt. T ist die globale Zeit. Insgesamt ist diese Veröffentlichung als lückenhaft (es fehlen zum Beispiel auch die Bilder und Beispiele 3 bis 8) und oberflächlich zu bezeichnen. So wird auch nur kurz erwähnt, dass ein Symbol F in der Grammatik für die Interaktion mit dem Vektor-Feld eingeführt wird. In Abhängigkeit von einem Modus-Flag wird damit der Turtle entweder an eine bestimmte Position gebunden oder er kann sich (basierend auf den Differenzialgleichungen) frei im Raum bewegen.

Es dürfte schnell klar werden, dass dies auch keine sonderlich intuitive oder einfache Art und Weise ist, die Bewegungen zu beschreiben und dann auch umzusetzen. Und zwar scheinen weder die Beschreibung der L-Systeme noch die Umsetzung und Modellierung der äusseren Kräfte sowie die zugrundeliegenden Berechnungen sonderlich benutzerfreundlich und effizienzt zu sein. Mal ganz davon abgesehen, dass sich reine L-Systeme nur bedingt gut für diesen Zwecke eignen, da einige dort nicht erfasste Parameter und Eigenschaften jeder einzelnen Pflanze auch in die Reaktion auf den Wind einfliessen sollten.

6.3 Maya

Als kleinen Exkurs wollen wir hier kurz auf das Programm Maya eingehen und welche Möglichkeiten für die Animation in diesem Programm vorgesehen sind. Laut Wikipedia: Maya ist eine professionelle und sehr verbreitete 3D-Visualisierungs- und Animationssoftware. Sie wird hauptsächlich in der Film- und Fernsehindustrie und bei der Erstellung von Grafiken für Computer- und Videospiele eingesetzt (zum Beispiel Herr der Ringe 3 oder Doom 3). Die Entwicklerfirma Alias wurde im Januar 2006 von einem ihrer größten Konkurrenten, Autodesk, übernommen.

Daneben wird Maya auch in anderen Bereichen verwendet, wie der Industriellen Fertigung, Architekturvisualisierung und in Entwicklung und Forschung. Maya ist damit eines der bekanntesten und meistgenutzten Softwareprodukte im Bereich 3D-Modellierung, Computeranimation und Rendering.

Die Modellierung von computergenerierten Pflanzen mit Maya geschieht prozedural, der Anwender hat verschiedene Möglichkeiten, wie er kleinere Grund-Objekte (zum Beispiel Zylinder oder selbstdefinierte Objekte) modelliert und muss dann dafür sorgen, dass diese an die richtigen Positionen gesetzt werden, damit sie im Gesamtbild die gewünschte Pflanze ergeben. Für die Animation von mit Maya erstellen Pflanzen und Szenen gibt es die folgenden Möglichkeiten (siehe auch [May06]):

- Path Animation
- Keyframe Animation
- Nonlinear Animation

In der Path Animation wird ein Objekt entlang eines Pfades bewegt, diese ändert keine Attribute und damit auch nicht das Aussehen der Objekte. Die Keyframe Animation kann auf fast jedes beliebige Attribut angewendet werden, hat aber wieder Nachteile wie sie in 4.7 schon genannt wurden. Nichtlineare Animation entspricht der Vorgabe von einigen Schlüsselwerten, die auf einer Kurve für die Attribut-Entwicklung liegen, und der nicht-linearen Interpolation dieser Werte (wie in Bild 6.2 zu sehen ist).

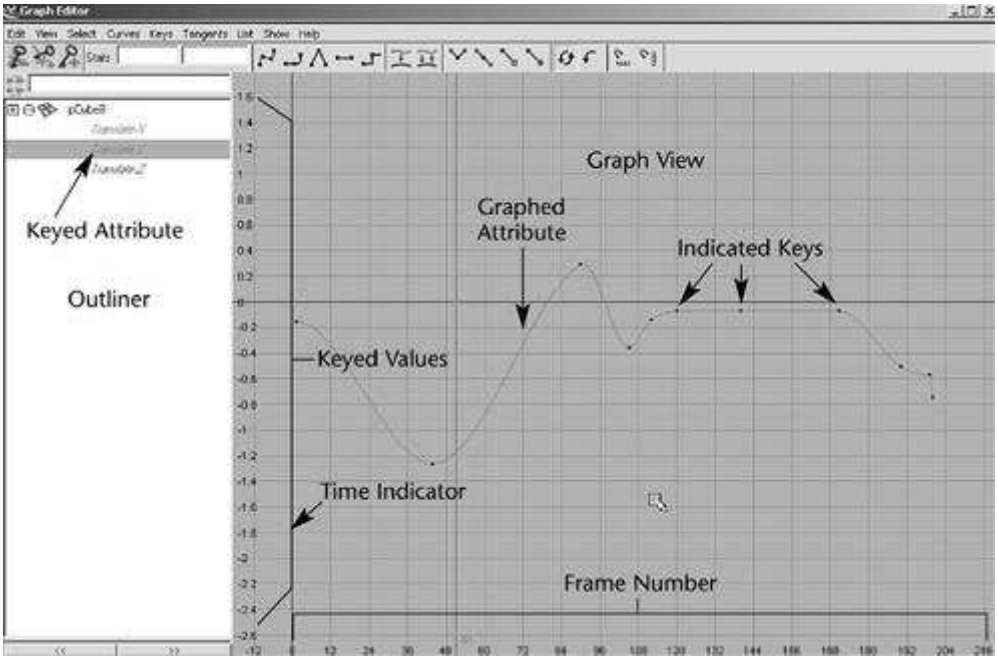


Abbildung 6.2: Nicht lineare Attribut-Entwicklung in Maya (aus [May06])

Auch hier muss aber alles von Hand gemacht werden. Das mag für eine einzelne Wachstums-Animation noch funktionieren, aber ist für die umfassende umwelt-getriebene Animation sicher noch keine praktikable Lösung. Bestenfalls kann man umwelt-getriebene Animation wie Wind hier durch ein zeitabhängiges grossflächiges Displacement modellieren, aber das würde insgesamt sicher zu unnatürlich (weil zu gleichmässig) aussehen.

6.4 Umwelt-getriebene Animation bei der regelbasierten Objekterzeugung

Der wesentliche Vorteil der regelbasierten Objekterzeugung für die umwelt-getriebene Animation ist die Trennung zwischen Geometrie und Topologie und natürlich überhaupt die Erzeugung der Geometrie in Verbindung mit möglichen Materialeigenschaften. Eigentlich braucht man für die Animation und insbesondere die Visualisierung der Animation beides, aber die wesentlichen Berechnungen finden in der Geometrie statt. Die Beschreibung der Topologie ist dann für das Visualisieren der berechneten Ergebnisse interessant. Da sich Bewegungen von in der Hierarchie innen liegenden Pflanzenelementen sehr stark auf weiter aussen liegende Pflanzenelemente auswirkt, ist gerade der modulare Aufbau der Pflanzen in Verbindung mit einer auf einem Szene-Graphen aufbauenden Visualisierungsumgebung besonders vielversprechend. Für die Visualisierung wird eine Pflanze aus einer Folge von abwechselnd Komponenten und lokalen Transformationen beschrieben. Ändert sich die Position eines dicken Astes, wird damit auch die Position eines an diesem dicken Ast befindlichen kleineren Astes geändert.

Das reine Visualisieren der umwelt-getriebenen Animation (nach der eigentlichen Berechnung) sollte damit von der Technik her keine grosse Herausforderung sein, es gibt allerdings andere wichtige Punkte oder Fragen zu beantworten:

1. Was wird genau modelliert und auf was wird Wert gelegt?
2. Wie werden die Bewegungen genau berechnet?
3. Was kann gemacht werden, um die umwelt-getriebene Animation effizient zu gestalten?

Mit diesen Punkten werden wir uns im Rest dieses Kapitels und auch in Kapitel 8 beschäftigen.

6.4.1 Wie und was wird modelliert

Die grundlegende Frage an dieser Stelle ist, was man Modellieren möchte und wie man dieses erreichen kann. Steht die physikalische Korrektheit oder ein möglichst natürliches Aussehen in Verbindung mit der Einfachkeit und Effizienz der Berechnungen im Vordergrund?

In [Sta97] stellt der Autor fest, dass es sehr schwer und umständlich wäre, den Effekt von Wind basierend nur auf reinen (benutzerdefinierten) Keyframes umzusetzen. Er stellt weiterhin fest, dass die resultierenden Bewegungen von Bäumen im Wind zufällig sind bzw. zufällig aussehen. Dies liegt an den unterschiedlichen Kräften, die von den vielen verschiedenenen Blättern herrühren und weil der Wind innerhalb eines Baumes nicht die gleiche Richtung beibehalten kann. Der Autor modelliert daher nur stochastisch/zufällig den Effekt des Windes und ist damit von den grundlegenden Berechnungen her sehr effizient.

Um die Animation dann umzusetzen, werden in der oben genannten Veröffentlichung zeitabhängige Folgen von Verschiebungswerten berechnet. Diese Folgen hängen von den Materialeigenschaften der Bäume und Äste ab und werden auf die Bäume angewendet. Zwischen den verschiedenen „time samples“ (die Zuordnung eines festen Wertes zu einem Zeitpunkt) innerhalb der Verschiebungen wird linear interpoliert. Durch das Einführen eines möglichen Verzögerungsfaktors kann sogar eine Szene mit vielen gleichen Bäumen und der gleichen Verschiebungs-Folge für alle diese Bäume benutzt werden, um einen ganzen Wald im Wind zu modellieren.

Wenn man bei der physikalischen Korrektheit Abstriche macht, dann scheint das eine gute Richtung zu sein, wobei in der Praxis sicherlich ein kombiniertes Modell ebenfalls gut geeignet ist. Also ein Modell, in dem die Bewegungen des Stammes und der Äste basierend auf dem Wind berechnet werden und für Blätter nur ein stochastisches/zufälliges Modell angewendet wird. Die Grenze zwischen dem Berechnen und dem stochastischen Modell muss dabei nicht notwendigerweise bei den Blättern liegen. Es ist möglich, dass die Bewegungen für den Stamm und die Äste nur nur bis zu einer gewissen Tiefe oder Durchmesser berechnet werden. Alle tieferliegenden Objekte und damit insbesondere auch Blätter fallen dann unter das stochastische Modell. Dadurch, dass für grössere Teile der Pflanzen ein stochastisches Modell benutzt wird, reicht es dann wahrscheinlich sogar aus, wenn eine einzige Windrichtung und Windstärke für die ganze Szene angegeben wird. Damit könnte Wind sehr einfach in die Szene eingefügt werden. Unser Vorschlag für das Modellieren des Windes ist, dass dieser zu verschiedenen Positionen innerhalb der Szene oder Welt mit einer unterschiedlichen Richtung angegeben werden kann. Gibt man ihn nur an einer Position an, so gilt diese Windrichtung in der ganzen Szene. Gibt man die Windrichtung und Windstärke an verschie-

denen Positionen an, so werden diese Werte für alle anderen Positionen in der Szene interpoliert. Mit einer solchen Umsetzung des Windes könnte dann auch der Effekt eines schwebenden Helikopters umgesetzt werden. Dazu müsste man die Windrichtung direkt unter dem Helikopter senkrecht auf den Boden richten, während die Windrichtungen zu den Seiten etwas abflacht. Dies ist in dem Bild 6.3 dargestellt. Der Helikopter befindet sich an der Position, die mit x markiert ist und der Betrachter schaut von der Seite auf den Helikopter.

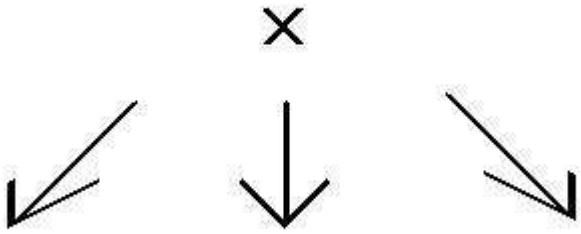


Abbildung 6.3: Windrichtungen bei einem Helikopter an der Position x

Ein mögliches Problem für die oben genannte Aufteilung der Berechnung in echte Berechnungen für die ersten Module und nur zufälligen Berechnungen für alle anderen Module wie Blätter, können allerdings Pflanzen sein, bei denen die Stämme bzw. Stengel im Vergleich zu den Blättern und Blüten relativ klein sind und damit dem Wind nur sehr wenig „Angriffsfläche“ bieten. Wo also die Beeinflussung durch den Wind eher daher rührt, dass dieser auf die Blätter oder Blüten und nicht auf den Stengel einwirkt. Hier wäre es sinnvoll, wenn man keinen zufälligen Anteil für diese Pflanzen in der Animation berücksichtigt (also auch für die Blätter und Blüten die Berechnungen durchführt) und wenn die Beeinflussungen von unterschiedlichen Objekten in der Hierarchie der Pflanze bidirektional laufen. Bidirektional bedeutet hier, dass die Positions-Änderungen aufgrund von Wind-Einfluss nicht nur bottom-up durch die Pflanze propagiert werden, sondern auch top-down ablaufen können. Die damit gewonnene Flexibilität bietet auch weitere Vorteile in den Möglichkeiten was man neben Wind mit dem Verfahren noch modellieren kann (siehe dazu Kapitel 9.2) und in Bezug auf die physikalische Korrektheit, die zwar nicht umfassend angestrebt wird, aber bis zu einem gewissen Punkt doch wünschenswert ist.

6.4.2 Berechnung und Umsetzung der Bewegung

In diesem Abschnitt geht es zuerst um das Berechnen der aus dem Wind resultierenden Bewegung und wie die Eigenschaften der Module darauf Einfluss nehmen. Danach werden wir untersuchen, wie die Bewegung im Rahmen des PlantAnimators mit affinen Transformationen umgesetzt werden kann. Wir unterscheiden hierbei, wie im vorherigen Abschnitt schon geschrieben, zwischen der auf dem Wind basierenden Bewegung und einer zufälligen Bewegung.

Um die auf dem Wind basierende Bewegung jetzt berechnen zu können, müssen wir uns zuerst einmal Gedanken machen, welche Faktoren Einfluss auf diese Bewegung nehmen sollen:

- Die Festigkeit des Materiales.
- Der Durchmesser des Pflanzenobjektes, für welches die Bewegung berechnet werden soll.
- Die Länge des Pflanzenobjektes, für welches die Bewegung berechnet werden soll.
- Die Position und Lage des betrachteten Pflanzenobjektes zum Wind.

Die daraus resultierende Bewegung sollte nicht nur eine einfache Verschiebung sein, sondern auch die Möglichkeit einer Rotation des Pflanzen-Elementes in sich selbst bieten. Eine Rotation wie sie zustande kommt, wenn der Wind auf zwei Seiten eines Astes ungleichmässig einwirkt, weil die eine Seite etwas mehr verdeckt wird. Ein Vorschlag wäre es, diese Rotation als zufällige Komponente mit wenig Spielraum zu implementieren, da sie auch nur eine sehr untergeordnete Rolle spielt.

Der vierte Parameter, also die Position und Lage, wäre insbesondere für Blätter oder Objekte mit ungleichmässiger Geometrie interessant. Hier hängt der Einfluss des Windes auf diese Objekte wesentlich von der Lage dieser zu ihm ab. Da Blätter hier aber nur zufällig animiert werden, hat dieser Parameter aber nicht ganz so viel Einfluss, wie man vermuten könnte.

Aus den ersten drei Werten kann dann die Wind-Beeinflussbarkeit des Modules berechnet werden. Der Benutzer gibt jedem Modul in der Grammatik einen Festigkeits-Wert vor. Dieser wird dann für die Berechnung der Wind-Beeinflussbarkeit noch mit Werten multipliziert, die durch die jeweils aktuellen Skalierungswerte für Breite und Länge beeinflusst werden. Die Wind-Beeinflussbarkeit sollte bei bei zunehmender Breite niedriger und bei zunehmender Länge grösser werden. Zu überlegen wäre, ob sie bei zunehmender Breite zu Beginn nicht erst etwas grösser

wird und ab einem gewissen Punkt dann erst niedriger wird. Dies würde zu Beginn mehr Gewicht auf das Vergrösseren der Angriffsfläche legen und erst ab einem gewissen Mindestwert würde die gesteigerte Stabilität greifen. Ein Vorteil dieses Verfahrens für unser System wäre, dass kein echter Durchmesser und insbesondere keine möglicherweise ungleichmässige Geometrie des Durchmessers benötigt wird. Diese Information kann durch den Benutzer in den Festigkeitswert mit einfliessen. Die Frage der tatsächlichen Berechnung der Wind-Beeinflussbarkeit verschieben wir auf das nächste Kapitel.

Ein weiterer Punkt, der in der Berechnung der Bewegung in einem Zeitschritt Einfluss nehmen sollte, wäre das Berücksichtigen einer alten Bewegung oder Ablenkung. Hört der Wind plötzlich auf oder wechselt die Richtung, so wird es mehr als nur ein Animations-Frame dauern, bis die Pflanze darauf reagiert hat. Daher sollte die letzte Ablenkung immer mit berücksichtigt werden.

Um auch bei der Darstellung der umwelt-getriebenen Animation in dem Bereich der affinen Transformationen zu bleiben, wäre es denkbar, dass die Bewegung der umwelt-getriebenen Animation mit einer Scherung umgesetzt wird. Das Benutzen einer affinen Transformation würde nicht den eingangs dieser Arbeit genannten Zielen der Quantisierung und SharedGeometry widersprechen. Ausserdem könnte eine weitere affine Transformationen leicht in den Szene-Graphen, der schon aufgrund der Pflanzen-Definition Translationen, Skalierungen und Rotationen enthält, eingefügt werden. Dazu müsste man dann die Richtung und die Stärke der Scherung bestimmen. Der Einsatz einer Scherung bedeutet auch gleichzeitig, dass die Beugung mit zunehmender Länge des Modules zunimmt, weswegen man diesen Aspekt sogar aus der Berechnung der Wind-Beeinflussbarkeit ausklammern kann.

Grundsätzlich kann eine Scherung im 3D-Raum wie im folgenden Bild 6.4 aussehen.

Die für die Scherung wichtigen Informationen sind also die Grundebene und zwei Faktoren (im Bild die Faktoren a und b) zur Beschreibung der Scherung. Als Grundebene nehmen wir in unserem Falle die Grundebene des Modules, die in dem lokalen Koordinatensystem des Modules einer Koordinatenebene nämlich der XZ-Ebene entspricht. Auf diese Ebene projezieren wir dann die Windrichtung. Wird das Modul als Teil einer Pflanze dargestellt, liegt diese Grundebene natürlich irgendwo im Raum. Dann muss diese Ebene während der Traversierung durch die Pflanzen-Module, die im Rahmen der Animation sowieso durchgeführt wird, für jedes Modul neu bestimmt werden. Dies kann durch eine Kombination aus der Grundebene des Vorgänger-Modules und der Ausrichtung des Modules an seinem Anknüpfpunkt geschehen. Die Ebene muss also bei dem Traversieren der Modulhierarchie fortwährend für das aktuelle Modul berechnet werden. Aus dem projezierten Bild der Windrichtung und der Stärke des Windes leiten wir

Die *Scherung in 3D* ist ebenfalls einfach darstellbar:

Eine Scherung parallel zur xy-Ebene um den Wert a in x-Richtung und den Wert b in y-Richtung erreicht man mittels

$$\begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

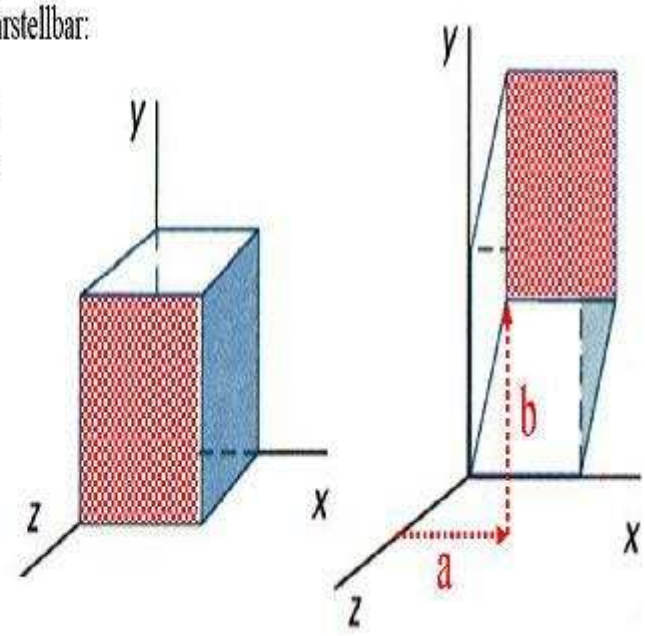


Abbildung 6.4: Scherung in 3D (aus [Pur07])

uns dann die beiden Faktoren a und b durch das Verhältnis der Winkel zwischen der projezierten Windrichtung und der beiden Koordinatenachsen her. Die Richtung der Projektion der Windrichtung bestimmt dazu noch das Vorzeichen der Faktoren a und b.

Die Gesamt-Stärke s für den Einfluss, den der Wind auf ein Modul nimmt, bestimmen wir mit folgender Formel:

$s = pwl * wb$

dabei ist

pwl Projezierte Windlänge: Wird allerdings noch mit einer gewünschten Maximalstärke für den Wind auf das Intervall 0 bis 1 skaliert

wb Wind-Beeinflussbarkeit: Berechnet sich aus Festigkeit des Modules in Verbindung mit Länge und Breite. Sollte bei zunehmender Breite niedriger und bei zunehmender Länge höher werden. Die genaue Berechnung dieses Faktors werden wir im nächsten Kapitel betrachten.

Der damit berechnete Wert s muss nun noch auf die beiden Faktoren a und b für die Scherung aufgeteilt werden. Wie dies funktioniert, wollen wir uns kurz auch

mit Bild 6.5 vor Augen führen. Dort sehen wir die Koordinatenachsen und zwei Beispiele für mögliche projezierte Windrichtungen.

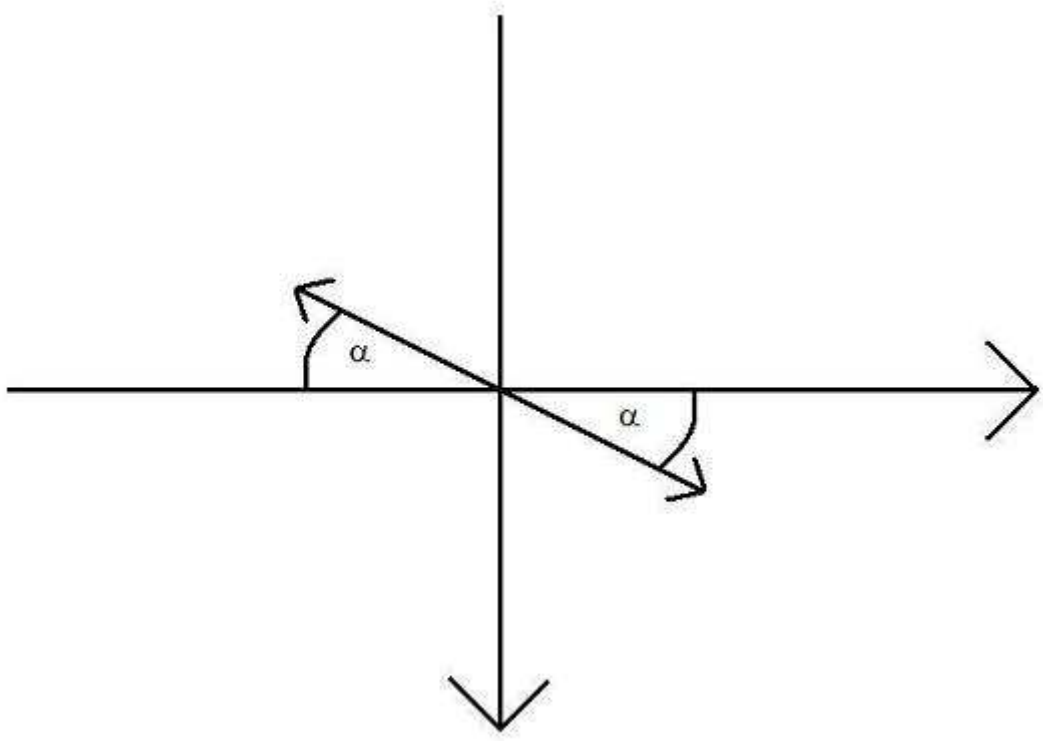


Abbildung 6.5: Berechnen der Faktoren a und b

α entspricht hier jeweils 30 Grad, der Winkel zu der anderen Koordinatenachse wäre damit 60 Grad. Die Berechnung von a und b sähe dann wie folgt aus:

$a = \frac{60}{90} * s, b = \frac{30}{90} * s$

Die Werte sind also für beide gezeigten Windrichtungen gleich, allerdings würde für die Windrichtung, die in dem linken oberen Quadranten liegt, jeweils noch ein Minuszeichen vor beide Faktoren eingefügt werden. Würde die Windrichtung in dem rechten oberen Quadranten liegen, dann würde nur b ein Minuszeichen bekommen, läge sie in dem linken unteren Quadranten, dann würde nur a ein Minuszeichen bekommen.

Die Veränderung in einem Animations-Schritt wird dann berechnet, indem man zuerst für a und für b die Differenz zwischen dem aktuell berechneten Wert und dem im letzten Animationsschritt tatsächlich eingenommenen Wert bestimmt. Von diesen Differenzen wird dann ein gewisser Anteil in diesem Animationsschritt ausgeführt. Dieser Anteil wird grösser, wenn die Windstärke zunimmt.

Die Scherung wird als affine Transformation in dem lokalen Koordinaten-System des Modules in den Szene-Graphen eingefügt, lediglich für die Projektion der Windrichtung auf die Grundebene muss die Grundebene im globalen Koordinaten-System bestimmt werden. Durch die Projektion der Windrichtung werden auch gleich Fälle behandelt, in denen der Wind sehr stark in Richtung der Ausrichtung eines Modules weht. Die Ausrichtung des Modules entspricht in seinem lokalen Koordinatensystem nämlich der Y-Achse, der Anteil der Windrichtung in Y-Achse wird aber durch die Projektion herausgefiltert. Eine solche Windrichtung könnte sonst bei dem Einsatz einer uneingeschränkten, also auch in Y-Richtung, Scherung zu einer ungewollten Verlängerung bei gleichbleibender Ausrichtung des Pflanzen-Modules führen.

Leider ist eine Scherung aber für die Schönheit der Darstellung nicht ganz so gut geeignet, da das Pflanzen-Modul zwar abgelenkt wird, aber das Endstück wird nicht weiter gebogen, sondern besitzt auch nach der Scherung noch die gleiche Ausrichtung. Dies sieht man in Bild 6.6. Ausserdem verlängert sich durch die Scherung die Länge des Modules, was man auch in Bild 6.6 sieht.

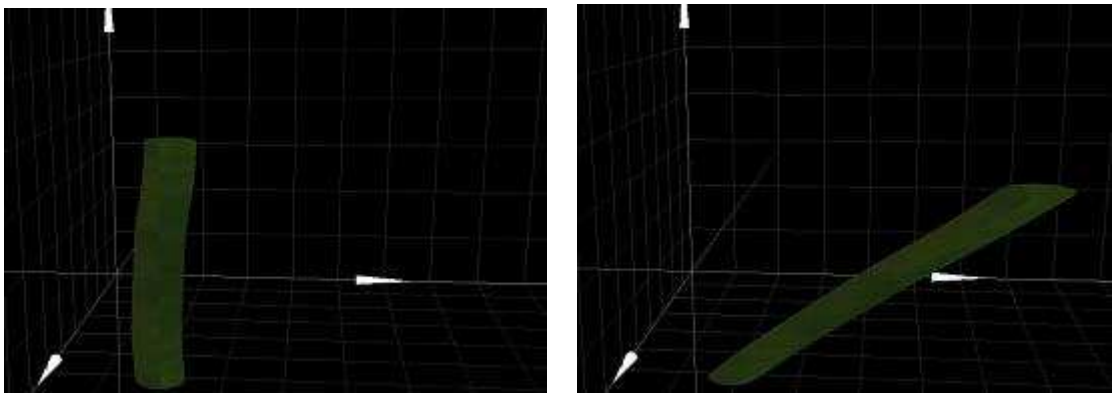


Abbildung 6.6: Ein einfaches Modul ohne und mit Scherung

Man sieht leicht, dass die durch die Scherung entstandene Verlängerung durch eine Skalierung rückgängig gemacht werden könnte. Dazu muss lediglich berechnet werden, um wieviel sich das Pflanzen-Modul verlängert hat. In dem Bild 6.7 sehen wir links die Aufsicht auf die XZ-Ebene von oben und rechts die Sicht auf die XY-Ebene aus der Z-Richtung. Wir setzen hier mal o.B.d.A. die Länge des Modules mit 1 fest, nach der Scherung um die Faktoren a und b ist das Modul dann verzogen. Die Länge der Diagonalen c in der XZ-Ebene (unterhalb des verzerrten Modules) lässt sich als $\sqrt{a^2 + b^2}$ berechnen, während der Abstand des Modul-Endes von der XZ-Ebene immer noch bei 1 liegt. Die neue Länge des Modules nach der Scherung lässt sich dann mit dem Satz das Phytagoras berechnen als $\sqrt{1 + a^2 + b^2}$ und es liesse sich mit einer gleichmässigen Skalierung (in allen drei

Richtungen) um den Faktor $\frac{1}{\sqrt{1+a^2+b^2}}$ nach der Scherung wieder die gewünschte Länge erzeugen.

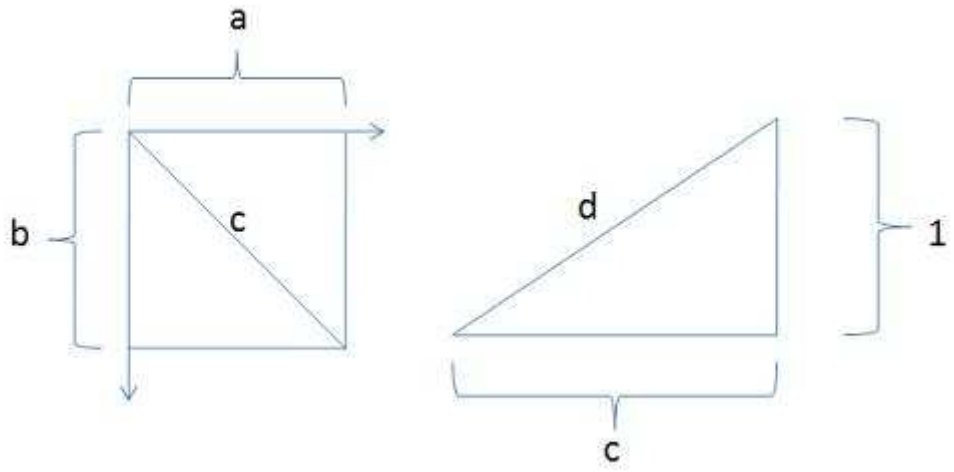


Abbildung 6.7: Berechnen der Modul-Verlängerung durch die Scherung

Die nächste Frage ist, ob diese Skalierung zur Korrektur der Länge nach oder vor der Scherung eingesetzt werden muss. Nach der Scherung muss man eine Skalierung in allen 3 Achsen machen, die als ungewünschtes Nebenprodukt dafür sorgt, dass das Pflanzenmodul dünner wird (wie man in Bild 6.8 auf der rechten Seite sieht). Macht man die Skalierung vor der Scherung, so muss die Skalierung lediglich in Y-Richtung gemacht werden und hat damit keinen Einfluss auf die Breite des Modules (wie man in Bild 6.8 auf der linken Seite sieht).

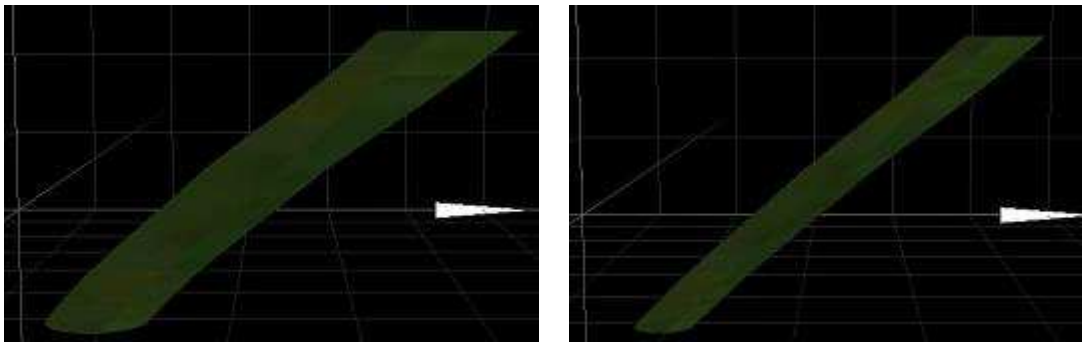


Abbildung 6.8: Das Anwenden der Skalierung vor und nach der Scherung

Das Benutzen einer Scherung scheint nicht ganz unproblematisch zu sein. Daher wollen wir kurz auf andere Möglichkeiten, die Beugung des Modules mit einer affinen Transformation zu realisieren ohne eine Scherung zu benutzen, eingehen. Man könnte hierbei entweder mit einer Rotation oder mit einer Translation zu arbeiten. Darstellungen zu diesen möglichen Ansätzen sehen wir in Bild 6.9. In dem Bild entspricht die erste Zeile einer Rotation des gesamten Modules um eine Achse, die der nicht eingezeichneten Z-Achse entspricht. Diese Vorgehensweise hat aber zwei Nachteile. Als erstes wird die Beugung eines Modules in der Praxis nicht als so gleichmässig empfunden, d. h. die Beugung sollte am Ende des Modules grösser sein als am Beginn. Zweitens wird neben der gewünschten Drehung der Endebene des Modules auch die Anfangsebene des Modules gedreht, was zu ungewünschten Nebeneffekten in Verbindung mit dem Vorgänger-Modul führen kann.

In der zweiten Zeile von Bild 6.9 sehen wir die interpolierte Anwendung einer Translation auf die einzelnen Vertices des Modules. Indem die Translation gegen Ende des Modules stärker zum Tragen kommt, wirkt die Beugung viel natürlicher, allerdings wird hier die Endebene nicht gedreht.

Die dritte Zeile in Bild 6.9 entspricht der interpolierten Anwendung einer Rotation auf die einzelnen Vertices des Modules. Diese sorgt für ein natürlicheres Aussehen der Beugung und einer Drehung der Endebene des Modules. Allerdings werden wir diese Idee im Rahmen dieser Arbeit nicht mehr weiterbetrachten. Sie baut sehr stark auf der Behandlung und Berechnung von einzelnen Vertices auf (was allerdings durch aktuelle Grafikkarten und GPUs gut unterstützt werden kann) und bietet damit Raum für weitere Forschungen.

Insgesamt zeigt sich, dass ein vertex-basierter Ansatz auch interessante Lösungsmöglichkeiten bietet. Ein solcher Ansatz würde nichts an der grundsätzlichen Modellierung von Pflanze und Umwelt ändern. Die Berechnungen zu dem Wind, also die Projektion des Windes auf die Grundebene eines Modules und das Berechnen der Stärke s, werden dabei gleich bleiben. Lediglich das Aufteilen in a und b würde man sich sparen können, da die Richtung von s beispielsweise gleich für das Bestimmen der Rotationsachse benutzt werden kann. Die Rotationsachse kann dann als Kreuzprodukt aus Ausrichtung des Modules und s berechnet werden. Allerdings muss man bei den vertex-basierten Ansätzen davon ausgehen, dass diese Probleme bei dem Nutzen von SharedGeometry- und Quantisierungs-Konzepten aufwerfen. Die Frage, ob es möglicherweise doch umsetzbar ist, einen vertex-basierten Ansatz so einzubauen, dass die anderen Optimierungs-Konzepte nicht negativ beeinflusst werden, verschieben wir in den Ausblick dieser Arbeit.

Für die zufällige Bewegung von Blättern und Modulen, die aufgrund ihrer Tiefe nicht voll animiert werden, kann auf die Projektion der Windrichtung auf die Grundebene des Modules verzichtet werden. Die Richtung der Scherung ist hier

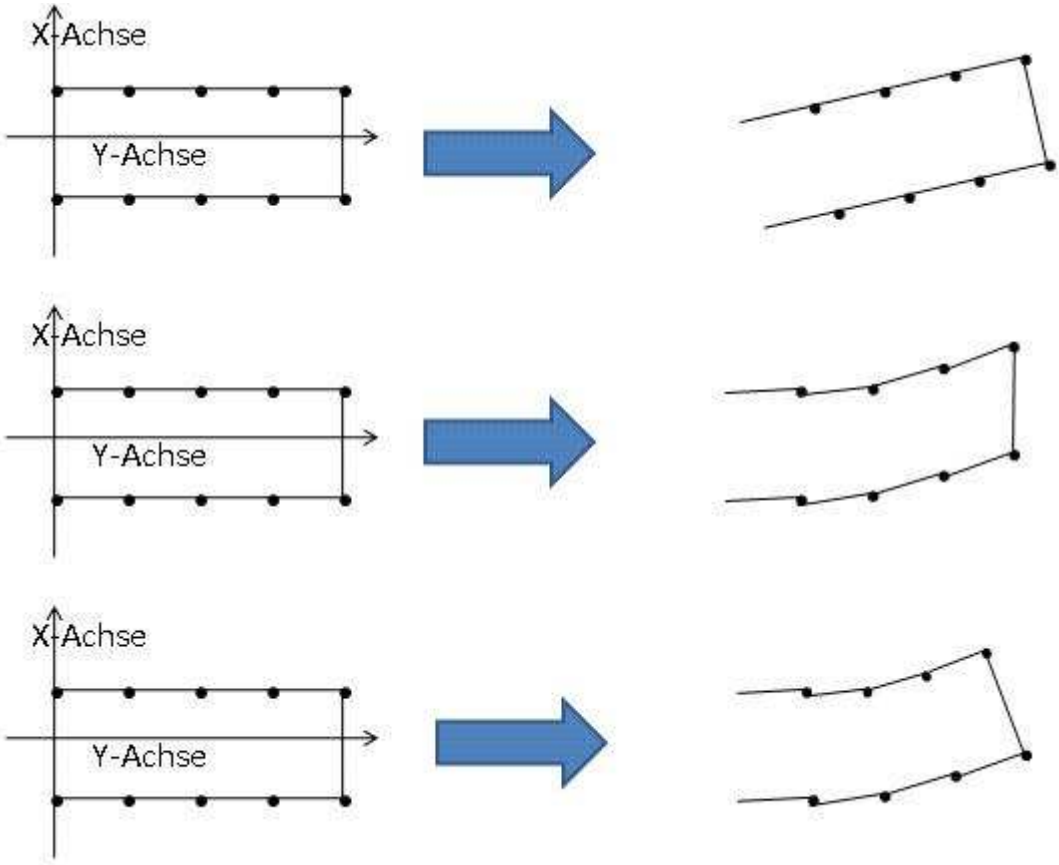


Abbildung 6.9: Mögliche Anwendung von affinen Transformationen auf Vertices eines Modules

nämlich zufällig, d.h. die Faktoren a und b ändern sich zeitabhängig und schrittweise. Je stärker der Wind ist (unabhängig von der Richtung) desto grösser die schrittweisen Änderungen dieser Faktoren.

Damit lässt sich also ein Mass für die Ablenkung berechnen. Eine noch offene Frage ist, ob man das Abbrechen von Pflanzenteilen während der Animation berücksichtigen möchte. Dazu könnte man entweder einen Pflanzen- oder Modulabhängigen Grenzwert definieren oder einen Wert aus anderen Parametern wie Festigkeit und Länge berechnen lassen, ab dem dieses „Unglück“ geschieht. Wobei ein solches Vorgehen weitere Probleme aufwirft, weil das abgebrochene Pflanzenteil entweder als neues und autonomes Objekt in der Szene entsteht oder weil das Pflanzenteil zwar abgeknickt wird, aber nicht abbricht. In letzterem Fall würde das Aussehen der Pflanze wesentlich beeinflusst werden und es wäre unklar, wie sich ein solcher Zweig im Rahmen der umwelt-getriebenen Animation verhält.

Eventuell könnten auch mehrere Scherungen benutzt werden, um beispielsweise

auch Gravitation umsetzen zu können. Bei der Gravitation bräuchte man noch ein Gewicht für jedes Modul, welches dann linear mit Länge und Breite multipliziert wird. Das Gewicht sollte in der Grammatik verankert sein. Die Aktualisierung des Gewichtes müsste dann eigentlich top-down geschehen, damit das Gewicht jederzeit aktuell ist. Erwartet man allerdings keine grossen oder sprunghaften Änderungen im Gewicht, so kann man das Gewicht auch nebenbei aktualisieren. Das bedeutet, dass in jedem Animationss-Schritt das Gewicht des aktuellen Modules als Summe der Gewichte der Kinder und damit auch aller Kindeskinder und so weiter und dem eigenen Gewicht bestimmt werden. Das bedeutet, dass das Gewicht je nach Tiefe der Modul-Hierarchie immer einige Schritte zu spät aktualisiert wird. Für die Lazy Animation (siehe Kapitel 8.3) müsste man dann sicherstellen, dass das Gewicht nach einer inaktiven Phase der Pflanze sofort wieder richtig berechnet wird.

Die nächste Frage an der Stelle ist, ob es in einem solches System noch Sinn macht, den Licht-Tropismus auch umzusetzen. Natürlich könnte man damit auch hier einen zusätzlichen Abweichungs-Wert für die Bewegung bestimmen, indem man die Position berechnet, die das Pflanzen-Modul aufgrund des Licht-Tropismus sicher gerne einnehmen würde und einen Teil der Differenz zwischen dieser Position und der aktuellen Position mit in die Animation hereinnimmt. Allerdings können wir davon ausgehen, dass bei dieser Umsetzung der umwelt-getriebenen Animation und sobald auch ein etwas stärkerer Wind vorhanden ist, der Einfluss des Lichtes untergehen wird.

Zusammenfassung dieses Abschnittes:

Für Module, die aufgrund des Tiefe-Parameters voll animiert werden, wird das Folgende ausgeführt:

1. Projektion der Windrichtung auf die aktuelle Grundebene des Modules
2. Berechnen von s
3. Bestimmen der gewünschten Scherungsfaktoren a und b
4. Bestimmen wieviel von dem Unterschied zwischen a_{alt} und a sowie zwischen b_{alt} und b in diesem Animations-Schritt umgesetzt wird (auch noch einmal abhängig von der Länge bzw. Stärke der projezierten Windrichtung)
5. Bestimmen einer möglichen kleinen Änderung in der inneren Rotation des Modules

Optional könnte noch das Bestimmen einer Ablenkung durch Gravitation hinzu kommen, dies entspräche einer Wiederholung der Schritte 1 bis 3 mit dem Gravitationsvektor anstatt der Windrichtung und unter Berechnung der Stärke s aus

Gewicht, Festigkeit des Materiales, Länge und Breite des Modules sowie Stärke der projezierten Gravitation. Für eine realistische Darstellung sollte die Gravitation allerdings vor dem Wind umgesetzt werden. Wahrscheinlich reicht es sogar aus, wenn die Einwirkung der Gravitation nicht in jedem Schritt aktualisiert wird.

Für Blätter oder Module, die aufgrund des Tiefe-Parameters nicht zu den voll animierten Modulen gehören, wird lediglich und zufällig ein kleiner Offset berechnet (dieser wird grösser, wenn der Wind stärker ist), der zu der momentanen Position addiert (oder subtrahiert wird). Auch diese Bewegung wird mit einer lokalen Scherung des Blattes bzw. Modules umgesetzt, nur dass die Faktoren a und b zu keinem Zeitpunkt tiefgehend berechnet werden.

Für beide Varianten wird es sicherlich sinnvoll sein, das maximale Mass der Scherung auf einen bestimmten Wert zu begrenzen, weil sonst wieder unglaubwürdige Ergebnisse herauskommen könnten.

6.4.3 Effiziente umwelt-getriebene Animation

Das Animieren einer einzelnen Pflanze sollte eigentlich keine grossen Effizienz-Probleme hervorrufen, wie sieht es aber aus, wenn viele Pflanzen oder eine ganze Pflanzenwelt animiert werden müssen? Auch dazu kann man sich einiges einfallen lassen.

Zu allererst sollten die grundlegenden Berechnungen effizient und daher am besten von einfacher Natur sein. Zusätzlich kann es sehr hilfreich sein, wenn man unnötige Berechnungen identifizieren kann. Letzteres geschieht unter anderem durch Abweichungen von der physikalischen Korrektheit (dass also nicht für alle Teile der Pflanze wirklich eine dedizierte Berechnung durchgeführt wird). Weitere Ideen für das Eliminieren von unnötigen Berechnungen können das Blickfeld und die Entfernung des Betrachters von der Pflanze ausnutzen. Die Ausnutzung des Blickfeldes für das Konzept der sogenannten ‘Lazy Animation‘ wird in Kapitel 8 genauer beschrieben, die Ausnutzung der Entfernung zwischen Pflanze und Betrachter ermöglicht so etwas wie Level of Detail für die Animation (auch darauf wird im gleichen Kapitel näher eingegangen).

Das Ziel der Lazy Animation ist es, in einer grossen virtuellen Welt diejenigen Pflanzen zu erkennen, die animiert werden müssen (die also Rechenzeit benötigen). Gleichwohl muss sichergestellt sein, dass auch bei schnellen Bewegungen des Betrachters keine Artefakte oder unnatürlichen Bewegungen bei den Pflanzen, die zwischenzeitlich nicht berechnet wurden und dann aktualisiert werden müssen, auftreten. Das heisst, wenn also eine Pflanze zu einem Zeitpunkt nicht aktiv war und sie zu einem späteren Zeitpunkt aktiv wird, dann muss es möglich sein, mit möglichst wenig Aufwand dafür zu sorgen, dass die Pflanze aktualisiert wird und

zwar sowohl im Sinne der wachstums-bedingten als auch der umwelt-getriebenen Animation. Die Ersparnis durch die Nicht-Animation muss dazu noch grösser sein als die Kosten für die fortwährende Berechnung des Blickfeldes und auch einer gewisse Vorhersage, welche Teile der Szene demnächst in das Blickfeld rücken werden, um diese frühzeitig aktivieren zu können.

Im Gegensatz zu der Lazy Animation entspricht Level of Detail für die Animation keinem „Alles oder Nichts"-Konzept. Sie kann mit der bekannten Level of Detail-Darstellung im Rahmen der Visualisierung (im Allgemeinen der entfernungsabhängigen Verringerung der geometrischen Komplexität von Objekten) verbunden aber auch unabhängig davon betrachtet werden. Level of Detail bietet sowohl für die wachstums-bedingte als auch die umwelt-getriebene Animation verschiedene Möglichkeiten. Diese reichen von dem kompletten Einstellen der Animation über das Berechnen der Parameter nicht mehr in jedem einzelnen Animations-Frame bis hin zu normaler umwelt-getriebener Animation aber ohne die zufällige Bewegung für einzelne Blätter.

Diese beiden Konzepte sind allerdings vom Modellieren der Pflanze relativ losgelöst und können lediglich unterstützt werden. Sie werden daher ausführlicher erst in Kapitel 8 beschrieben.

Es gibt aber noch weitere Möglichkeiten, wie die Effizienz der umwelt-getriebenen Animation basierend auf den im vorherigen Abschnitt vorgestellten Überlegungen verbessert werden kann.

So kann man auf zuerst auf die Idee kommen, dass man sich bei längerem und (fast) gleichbleibendem Wind die wiederholte Projektion der Windrichtung auf die Grundebene der Module sparen kann und wahrscheinlich dann auch die Faktoren a und b nicht in jedem Schritt neu berechnen muss. Darauf aufbauend stellt sich die Frage, ob es auch bei nicht ganz so gleichbleibendem Wind in jedem Animationsschritt nötig ist, die Wind-Berechnungen (Projektion und Faktor-Berechnung) durchzuführen. Würde man dieses nur in jedem n. Schritt machen und würde man den Wind auch nur so modellieren, dass dieser sich auch nur in diesen Schritten ändert, dann könnte man sehr viele Berechnungen sparen. Man verwendet einfach die im Schritt x berechneten Werte für a und b auch in den Schritten $x+1, x+2, ...x+(n-1)$. Dies sollte möglich sein, da in jedem Schritt auch nur ein Teil der Differenz von a_{alt} und a sowie b_{alt} und b im Rahmen der Animation umgesetzt wird.

Eine weitere Idee geht der Frage nach, ob man sich die Projektion des Windes auf die Grundebene nicht sparen kann, also der Frage, ob man aus der für das Vater-Modul projezierten Windrichtung und der Ausrichtung am Anknüpfpunkt nicht die projezierte Windrichtung für das Kind ohne aufwendige Projektion berechnen kann. Mit einer solchen Berechnung kann man sich sowohl das Bestimmen der

aktuellen Grundebene eines Modules als auch die Projektion des Windes auf diese Grundebene sparen. Betrachten wir dazu mal das Bild 6.10, in diesem Bild ist w der Wind, a die Projektion des Windes auf die Grundebene des Modules A und d die Ausrichtung des Modules B an dem Anknüpfpunkt von Modul A. Die Frage ist, ob die gewünschte Projektion des Windes auf die Grundebene von B (in dem Bild als b bezeichnet) sich nicht aus a und d berechnen lässt.

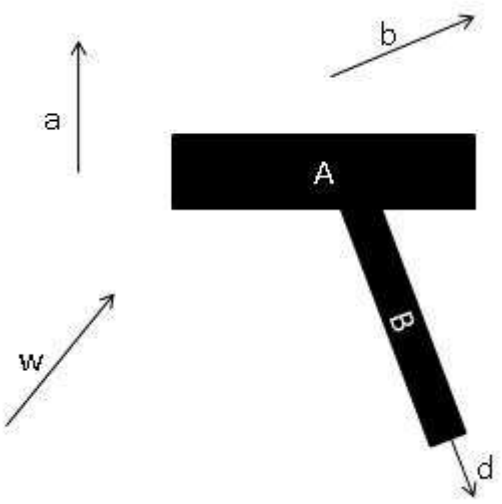


Abbildung 6.10: Berechnung des Windes für ein Modul ohne Projektion

Für b müssen wir dann sowohl seine Richtung als auch seine Länge berechnen können und das geht so:

Für die Richtung bilden wir zuerst den Vektor k als Kreuzprodukt aus der (im Bild nicht vorhandenen) Ausrichtung des Modules A und dem Vektor d (der Ausrichtung des Modules B). Wenn wir nun den Vektor a um den Vektor k rotieren lassen und zwar in dem Winkel, der zwischen den beiden Vektoren liegt, aus denen wir das Kreuzprodukt gebildet haben, dann haben wir die Richtung von b. Die Länge von b kann man berechnen, indem man den Winkel α zwischen d und w bestimmt und die Länge von d dann wie folgt berechnet $|d| = |w| * cos(90 - \alpha)$.

Mit dieser Vorgehensweise muss also nur am Anfang der rekursiven Aktualisierung einer Pflanze einmal die Windrichtung auf die Grundebene (die meistens der globalen XZ-Ebene entspricht) des allerersten Modules projeziert werden und diese Windrichtung kann dann für jedes Kind-Element ohne weitere Projektion neu berechnet werden.

Kapitel 7

Erweiterungen des PlantAnimator

Wie wir in Kapitel 5.5 und auch in Kapitel 6 gesehen haben, gibt es noch verschiedene Möglichkeiten, die erste Version des PlantAnimators zu verbessern und zu erweitern. Dies beinhaltet das Behandeln der umwelt-getriebenen Animation, welches bisher für computergenerierte Pflanzen selten in bestehende Systeme integriert wurde. Daneben handelt es sich um überwiegend praktische Probleme wie das Modellieren der Zeitachse oder die Übergänge in Bezug auf Geometrie und Durchmesser zwischen Pflanzen-Modulen, die innerhalb einer Pflanze aufeinander folgen.

Im Folgenden wollen wir uns ein paar der Punkte genauer anschauen, während andere Punkte aus Kapitel 5.5 wie beispielsweise die Integration einer Kollisionserkennung hier nicht weiter betrachtet werden. Dabei werden wir auch versuchen, sowohl die Wiederverwendbarkeit der Module zu erhöhen, als auch weiterhin die einfache aber auch sehr flexible Modellierweise der Modul-Geometrie zu erhalten.

7.1 Kontrolle der Zeitachse

Die Kontrolle der Zeitachse, also das Steuern und Abstimmen der Entwicklung von Parametern und Kindern der unterschiedlichen Module, ist wesentlich für das Ergebnis des PlantAnimators. Es muss hierbei darauf geachtet werden, dass die Modellierung der Zeitachse weder die insgesamt einfache und intuitive Modellierung durch den PlantAnimator noch die Wiederverwendbarkeit der Module beeinträchtigt. In dem ersten Ansatz des PlantAnimators gab es eine Verbesserungsmöglichkeit bei dem Querbezug zwischen der Zeitachse und der Wieder-

verwendbarkeit der Module. Ausserdem gab es dort noch keine Möglichkeit, die Länge der Animation insgesamt angeben zu können, und damit die tatsächliche Entwicklungsgeschwindigkeit einer Pflanze entweder beschleunigen oder verlangsamen zu können. Letztgenanntes sollte die Wiederverwendbarkeit der Module ebenfalls steigern, indem die Wiederverwendbarkeit von ganzen Pflanzen verbessert wird. In dem ersten Ansatz ergab sich die Länge der Animation und damit die Entwicklungsgeschwindigkeit aus den in den einzelnen Modulen gespeicherten Zeiten. Wollte man dann eine sich langsamer oder schneller entwickelnde Pflanze haben, so musste man Änderungen in den Modulen vornehmen.

Im Folgenden wollen wir einige der möglichen Probleme hervorheben, um uns danach mit den Lösungsmöglichkeiten zu diesen Problemen zu beschäftigen. In der ersten Version des PlantAnimators konnte das Folgende entweder nur durch das Ausnutzen von Features oder gar nicht umgesetzt werden.

Eine gegebene oder gewünschte Länge bzw. Dauer der Animation lässt sich mit einem Durchgang durch die Pflanze und der Analyse der dort verwendeten Lebens-Zeiträume umsetzen. Danach hat man die Zeit, die die Entwicklung der Pflanze (höchstens) brauchen wird. Diese könnte dann an die gewünschte Zeit angepasst werden, indem man nur die (virtuellen) zeitlichen Abstände der Aktualisierungsaufrufe ändert und damit die Zeit ausserhalb der Pflanze streckt oder staucht.

Die Koordinierung und Abstimmung des Wachstumes von verschiedenen Modulen innerhalb einer Pflanze wurde in der ersten Version des PlantAnimators durch das Ausnutzen von Features gelöst. Dies beinhaltet das Einführen von zusätzlichen Modulen mit speziell angepassten zeitlichen Parametern bei gleicher Geometrie und gleichen anderen Parametern. Das spricht dafür, dass zu überlegen ist, ob die Zeitachse nicht besser in die Grammatik eingearbeitet werden sollte. Dies würde einerseits die Notwendigkeit von zusätzlichen Modulen, die sich nur in den zeitlichen Parametern von anderen Modulen unterscheiden, eliminieren und andererseits auch die Wiederverwendbarkeit von Modulen erheblich erhöhen.

Zuerst wollen wir einmal betrachten, welche Koordinierungs-Probleme auftreten können und wie diese grundsätzlich gelöst werden könnten:

- Ein Parameter (wie beispielsweise der Winkel) soll sich erst später entwickeln als der Rest des Modules.
  - Die Steuerung könnte über eine sehr angepasste Funktion (die beispielsweise vorne lange auf dem Anfangswert verharrt) geschehen. Allerdings ist dann unklar, wann genau es mit der Parameteretwicklung losgeht, wegen der internen Streckung der Parameterentwicklungszeit auf die Lebenszeit des Modules.
  - Die Steuerung könnte über eine normale Funktion und einem Verzöge-

rungswert in der Grammatik geschehen. Aufgrund des im oberen Punktes beschriebenen Abstimmungs-Problemes bei der Streckung der Parameterentwicklungszeit auf die Lebenszeit des Modules (und der möglichen Streckung oder Stauchung der Lebenszeit einer Pflanze auf die komplette Animationszeit) sollte der Verzögerungswert am Besten wohl relativ und nicht absolut angegeben werden.

  - Die Steuerung über ein spezielles Modul. Man könnte ein weiteres Modul mit der gleichen Geometrie anlegen, welches als Kind nur das echte Modul am Ende seiner Entwicklung erzeugt und dann verschwindet. Die End-Parameter des Klones und Anfangs-Parameter des echten Modules müssten dann übereinstimmen, der verzögerte Parameter entwickelt sich im ersten Modul gar nicht und dann erst im zweiten, die Verzögerung enstpricht insgesamt der Life-Time des ersten Modules

- Ein Teil der Pflanze soll erst später oder zu einem ganz bestimmten Zeitpunkt erzeugt werden. Oder es soll auch die Reihenfolge des Wachstumes von verschiedenen Kindern des gleichen Modules aufeinander abgestimmt werden, also nicht nur jedes einzelne Kind soll irgendwann innerhalb des Breeding Intervalles entstehen, sondern das Entstehen verschiedener Kinder soll auch noch koordiniert werden.

  - Auch dies kann mit duplizierten Modulen bei angepassten Parametern gelöst werden. Soll einfach nur ein Teil der Pflanze verspätet entstehen, so wird ein sich nicht entwickelndes Modul angelegt, welches am Ende seiner Lebenszeit durch das echte Modul ersetzt wird. Soll das Entstehen von mehreren Kindern (o.B.d.A. x Kindern) koordiniert werden, so müsste man x weitere Module erzeugen, jedes der x weiteren Module erzeugt eines der Kinder (in der gewünschten Reihenfolge) und wird dann durch das nächste der duplizierten Module ersetzt.
  - Die Steuerung der Entstehung der Kinder kann aber auch über die Angabe von bestimmten Erzeugungs-Punkten in der Grammatik geschehen. Diese sollten aber wieder relativ zur Lebenszeit des Modules sein.

Beide Probleme lassen sich also sowohl mit einem (oder mehreren) zusätzlichem Modul mit anderer Zeitsteuerung (was bisher ja auch im PlantAnimator gemacht wurde) als auch durch das Einarbeiten der Zeitachse in die Grammatik lösen. Von daher wird es nun das Ziel sein, dass die Zeitachse in die Grammatik eingearbeitet werden kann. Fassen wir hier noch einmal zusammen, was dann alles in der Grammatik benötigt wird:

1. Die Lebenszeit zu jedem Modul und auch was nach dieser Lebenszeit passiert. Module können am Ende ihrer Lebenszeit bestehen bleiben, verschwinden oder durch andere Module ersetzt werden.

2. Relative Entstehzeitpunkte oder Zeiträume zu jedem Kind in der Grammatik, um das Entstehen der Kinder abstimmen zu können.

3. Mögliche relative Verzögerungswerte für die Parameterentwicklung. Je ein Wert für vor der Parameterentwicklung und ein Wert für nach der Parameterentwicklung.

Diese Sachen lassen sich ohne grosse Probleme auf das bisherige interne Konzept des PlantAnimators übertragen. Im Falle der Verzögerungswerte in der Grammatik werden diese einfach der Funktion mitgegeben (siehe auch Kapitel 5.1.2), die dann dafür sorgt, dass vor Beginn der eigentlichen Parameterentwicklung der Startwert und nach der eigentlichen Parameterentwicklung der End-Wert ausgegeben wird. Die relativen Entstehzeitpunkt oder Zeiträume werden durch den Scheduler umgesetzt. Dieser analysiert sowieso schon die Regeln der Grammatik und die Eigenschaften der beteiligten Module. Wo bisher beim Anlegen eines Modules zufällig bestimmt wurde, welche Kinder wann erzeugt werden, wird dieses aufbauend auf den Werten in der Grammatik nun nicht mehr ganz so zufällig durchgeführt. Das gleiche gilt auch für die Lebenszeit und der Festlegung, was nach der Lebenszeit mit einem Modul passiert. Bisher hat der Scheduler diese Informationen aus den beteiligten Modulen genommen, nun bekommt er sie aus der Grammatik.

Wir sehen also, dass diese Änderungen keine grossen Einschnitte in das bestehende System bedeuten, während die Möglichkeiten der Modellierung und der Animations-Steuerung wesentlich verbessert werden.

Bliebe noch die Steuerung der Gesamt-Animationszeit zu betrachten. Nachdem die Zeitachse nun in der Grammatik angegeben ist, kann durch einen einfachen Durchgang durch die Grammatik (ohne die Module laden zu müssen) bestimmt werden, wieviel Zeit die gesamte Animation bzw. Entstehung der modellierten Pflanze mindestens und höchstens dauern wird. Zwischen diesen Werten muss unterschieden werden, wenn man in der Grammatik Zeiträume für das mögliche

Entstehen von Kindern angibt. Diese beiden Werte müssen dann in Verbindung gebracht werden mit der Zeit, die für das gesamte Wachstum der Pflanze vorgesehen ist. Ohne den Unterschied in Mindest- und Höchstdauer wäre es recht einfach, da der Scheduler das Aktualisieren der Pflanze in virtuell kleineren oder grösseren Zeitschritten anstossen könnte und damit den Zeitverlauf ausserhalb der Pflanze entweder beschleunigt oder verlangsamt. Wenn aber zwischen Mindest- und Höchstdauer unterschieden werden muss, dann gibt es keine einfache und völlig korrekte Lösung wie man diese beiden Zeiten zu einer vorgegebenen Zeit ins Verhältnis setzen kann. Für diesen Zweck muss entweder der Benutzer vorgeben, ob seine Zeit-Angabe mit der Mindest- oder der Höchstdauer in Verbindung gesetzt werden soll, oder es muss die Mitte zwischen Mindest- und Höchstdauer genommen werden, um zumindest ungefähr das Ziel zu erreichen. Die Vorgabe durch den Benutzer würde dann bedeuten, dass der Scheduler an jedem Punkt, wo ein Zeitraum für das mögliche Entstehen von Kindern angegeben ist, sich entweder immer für die untere oder die obere Schranke des Zeitraumes entscheiden muss (je nachdem, welche Schranke der Benutzer gewählt hat). Das Benutzen von dem Mittelwert zwischen Mindest- und Höchstdauer basiert darauf, dass der Unterschied der beiden Zeiten nur durch die Angabe von Zeiträumen für das Entstehen von Kindern entsteht und innerhalb dieser Intervalle jedes Mal zufällig bestimmt wird, wann das Kind entsteht.

Wobei es nicht unbedingt passieren muss, dass die Angabe von Zeiträumen für das Entstehen von Kindern auch wirklich zu einem Unterschied in Mindest- und Höchstdauer führt. Würden wir zum Beispiel eine Blume modellieren, deren Seitenzweige mit ein bisschen Zufall bei den Zeitpunkten des Entstehens erzeugt werden, aber wo das Wachstum der Blüte länger dauert als das Wachstum jedes anderen Seitentriebes, dann könnte die Mindest- und Höchstdauer trotzdem die gleiche sein. Mit einem Durchlauf durch die ganze Grammatik (zum Beispiel beim Speichern) und dem Betrachten aller Wege von dem Startobjekt zu allen Blättern in der Modul-Hierarchie kann die Mindest- und Höchstdauer bestimmt werden. Stimmen diese nicht überein, kann der Benutzer bei der Angabe der gesamten Animationszeit darum gebeten werden, dass er sich für eine der beiden Zeiten entscheidet oder er kann entscheiden, ob er mit einer möglichen Abweichung von seiner angegebenen Gesamt-Animationszeit leben kann.

Eine weitergehende Frage an der Stelle wäre die Frage nach dem Zusammenhang zwischen der Pflanzenentwicklung und den Jahreszeiten. Wir haben in Kapitel 2.1.2 schon die sogenannte Thermonastie kennengelernt, also das mögliche Öffnen von Blüten basierend auf der äusseren Temperatur und wissen auch, dass viele Bäume im Herbst und Winter ihre Blätter verlieren.

Das würde bedeuten, dass sowohl Parameterentwicklungen (für das Öffnen der Blüten) als auch das Anwenden von Regeln in der Grammatik (für das Verlieren von Blättern) durch solche äusseren Faktoren gesteuert werden müssten. Diese

Frage wollen wir allerdings in den Ausblick verschieben. Es wird an anderer Stelle zu untersuchen sein, ob vielleicht das Steuern der Parameterentwicklung schon ausreicht, um alle gewünschten Effekte zu realisieren. Zumindest das Verlieren der Blätter kann mit sehr kleinen Werten für die Länge und Breite der Blätter realisiert werden und dem gleichzeitigem Einfügen eines losgelösten Blattes an genau der Stelle der Szene, welches dann von den wirkenden Kräften beeinflusst und umhergeweht wird.

7.2 Änderung der Breite und der Verjüngungsfaktor

In der ersten Version des PlantAnimators sollte die Breite von Modulen über einen Parameterwert in der Grammatik und Verjüngungsfaktoren bestimmt werden. Die Verjüngungsfaktoren sollten angeben, wie sehr die Breite (oder besser der Durchmesser) in Bezug auf die Länge des Modules abnimmt, der sich entwickelnde Parameter den Anfangs- und Endwert für den Durchmesser am Beginn des Modules.

Allerdings zeigten sich schon nach kurzer Zeit zwei mögliche Probleme bei der korrekten Umsetzung dieser Faktoren.

1. Der Anfangsdurchmesser eines Modules als Fortführung eines anderen Modules an einem TOP-Anknüpfpunkt sollte nach Möglichkeit zu dem Enddurchmesser des Vorgänger-Modules passen.
2. Wie wird der Verjüngungsfaktor eigentlich umgesetzt?

Zu Punkt 2 lässt sich sagen, dass der hier sogenannte bzw. im PlantAnimator bisher angegebene Verjüngungsfaktor eigentlich eine Kombination aus einem Faktor und einem Funktionsalias ist. Der Faktor soll angeben, um wieviel sich der End-Durchmesser vom Anfangs-Durchmesser unterscheiden soll, das Funktionsalias soll angeben, wie sich der Durchmesser zwischen diesen beiden Werten bzw. zwischen Anfang und Ende des Modules entwickelt. Die Umsetzung des Verjüngungsfaktores im PlantAnimator und deswegen mit affinen Transformationen ist allerdings nicht unproblematisch, für eine lineare Verjüngung könnte eine relativ einfache Scherung eingesetzt werden. Für eine andere Art der Verjüngung wird das Umsetzen dieser mit einer affinen Transformation schwierig, weil die Werte (oder mindestens einer) der Transformation nicht linear von dem y-Wert innerhalb des Modules abhängen müsste.

Es sieht daher so aus, als wenn eine dedizierte Verjüngung der Module (d.h. eine Verjüngung, die zusätzlich zu der Geometrie der Module definiert wird) nur über eine vertex-basierte Methode wirklich Sinn macht. Eine vertex-basierte Methode würde es dann auch erlauben, dass eine Funktion des Funktionseditors eingesetzt werden kann, um die tatsächliche Verjüngung in Abhängigkeit von dem y-Wert zu berechnen, weil dann ja die Kombination aus Startwert, Endwert und Verlauf bekannt ist. Der y-Wert würde dann die Rolle der Zeitachse im Funktionseditor übernehmen.

Unabhängig von diesem Problem ist allerdings der Punkt 1, also die Bestimmung des Anfangsdurchmesser eines Modules als Fortführung eines anderen Modules. Hier wäre es wünschenswert, wenn die Übergänge zwischen den Modulen möglichst glatt wären. Ein Problem an der Stelle ist allerdings der sehr offene Geometrie-Ansatz des PlantAnimators. Module werden zwar über Vertices definiert, aber es werden nur zwei Vertices zur Bestimmung des Startpunktes und der Ausrichtung hervorgehoben. Wollte man jetzt den Durchmesser bestimmen, so müsste man wissen, welche Vertices zusammen überhaupt Querschnitte (vorzugsweise am Beginn und Ende) des Modules bilden.

Möchte man dem Benutzer nicht aufbürden, dass er mindestens noch zwei Querschnitte durch jedes Modul angibt, dann wird man eine andere Lösung für das Bestimmen dieser Vertices finden müssen. Da grundsätzlich der echte Durchmesser auch für die umwelt-getriebene Animation von Interesse ist, scheint sich sogar ein komplizierter aber automatischer Ansatz zu lohnen, solange es sich um eine einmalig durchzuführende Vorverarbeitung handelt.

Im folgenden sind also zwei Punkte für uns interessant:

1. Wie kommen wir an die Vertices für je einen Querschnitt am Beginn und an den Übergangspunkten zu den nächsten Modulen?
2. Wie können diese Vertices dann benutzt werden?

Die erste Frage ist relativ einfach geklärt, die Vertices für die Querschnitte können zum Beispiel bestimmt werden, indem man ein Raycasting-Verfahren einsetzt. Die Geometrie der Module wird auf das y-Intervall $[0, 1]$ skaliert und die Ausrichtung sollte in y-Richtung liegen. Die Vertices für die Querschnitte können also als Vorverarbeitungsschritt bestimmt werden, indem bei dem Anlegen eines Modules verschiedene Strahlen am Beginn des Modules und für jeden TOP-Anknüpfpunkt (jeweils +- einem Epsilon) verfolgt werden und die Schnittpunkte zwischen den Strahlen und dem Modul bestimmt werden. Je geringer der Abstand zwischen den verfolgten Strahlen, desto genauer wird der ermittelte Querschnitt. Dabei müssen lediglich so viele Ebenen für das Modul mit Raycasting abgedeckt werden, wie es

TOP-Anknüpfpunkte gibt plus eine Ebene für den Start. Die Lage der Ebene für das Raycasting zu jedem TOP-Anknüpfpunkt wird durch die Lage des Punktes selbst und seiner Normalen bestimmt.

Würde man das Raycasting-Verfahren etwas ausweiten, dann könnte man damit auch ungleichmässige Geometrie wie zum Beispiel bei Blättern erkennen. Dazu müsste man allerdings nicht nur in den oben beschriebenen Ebenen Strahlen durch die Szene schicken und diese auf Schnittpunkte prüfen. Ungleichmässige Geometrie könnte dann bei dem Einfluss von umwelt-getriebener Animation berücksichtigt werden. Dies gilt auch für den Teil der umwelt-getriebenen Animation, der nur zufällig bestimmt wird, weil dort dann die Schrittweite der beiden Faktoren a und b unabhängig voneinander und an die Geometrie angepasst werden kann. Bietet ein Modul in x-Richtung mehr Angriffsfläche, dann werden die Schritte für den Faktor a grösser sein als die Schritte für den Faktor b.

Die so mit Raycasting bestimmten Vertices können dann für die Übergange benutzt werden. Zu allererst kann aus den Vertices der Durchmesser berechnet werden und an einem TOP-Übergangspunkt von einem Modul zum nächsten sollte dann der Enddurchmesser der Vorgänger-Modules in etwa dem Anfangsdurchmesser des Nachfolger-Modules entsprechen. Um das zu erreichen, kann dann die Skalierung für das nachfolgende Modul aufgrund der Werte für die Durchmesser und nicht durch den in der Grammatik gegebenen Wert bestimmt werden.

Leider bedeutet gleicher oder zumindest annähernd gleicher Durchmesser noch nicht auch gleiche Geometrie der beiden Module an den Übergangspunkten. Daher erscheint es ebenfalls praktikabel für den Übergang noch ein kleines Zwischenobjekt basierend auf den Vertices zu bestimmen. Dieses Zwischenobjekt kann durch einfache Interpolation (bei gleicher Anzahl an Vertices) oder ein einfaches Morphing-Verfahren bestimmt werden.

Für die Bestimmung des Zwischenobjektes scheiden volumenbasierte Morphing-Ansätze und Morphing über Feature Lines aus, da das Volumen nicht berücksichtigt wird (und auch nicht wichtig ist) und weil das Benutzen von Feature Lines die Interaktion des Benutzers zur Eingabe dieser Feature Lines benötigt.

Viele andere Morphing-Ansätze bauen darauf auf, dass eine gleiche Vertex-Edge-Topology existiert oder erzeugt wird, die dann zur Interpolation benutzt wird. Dies könnte zum Beispiel durch die Projektion der Objekte auf eine Sphere, das Verbinden der beiden Geometrien und das Rückprojezieren der neuen Geometrie auf beide Objekte geschehen.

Naheliegend scheint hier ein Ansatz über sogenannte star shaped Polygone. Ein Polygon nennt sich star shaped, wenn es einen Punkt gibt, von dem aus alle anderen Punkte des Polygones über eine gerade Linie, die ganz innerhalb des Polygones liegt, erreichbar sind. In diesem Falle könnte man jeweils Paare von

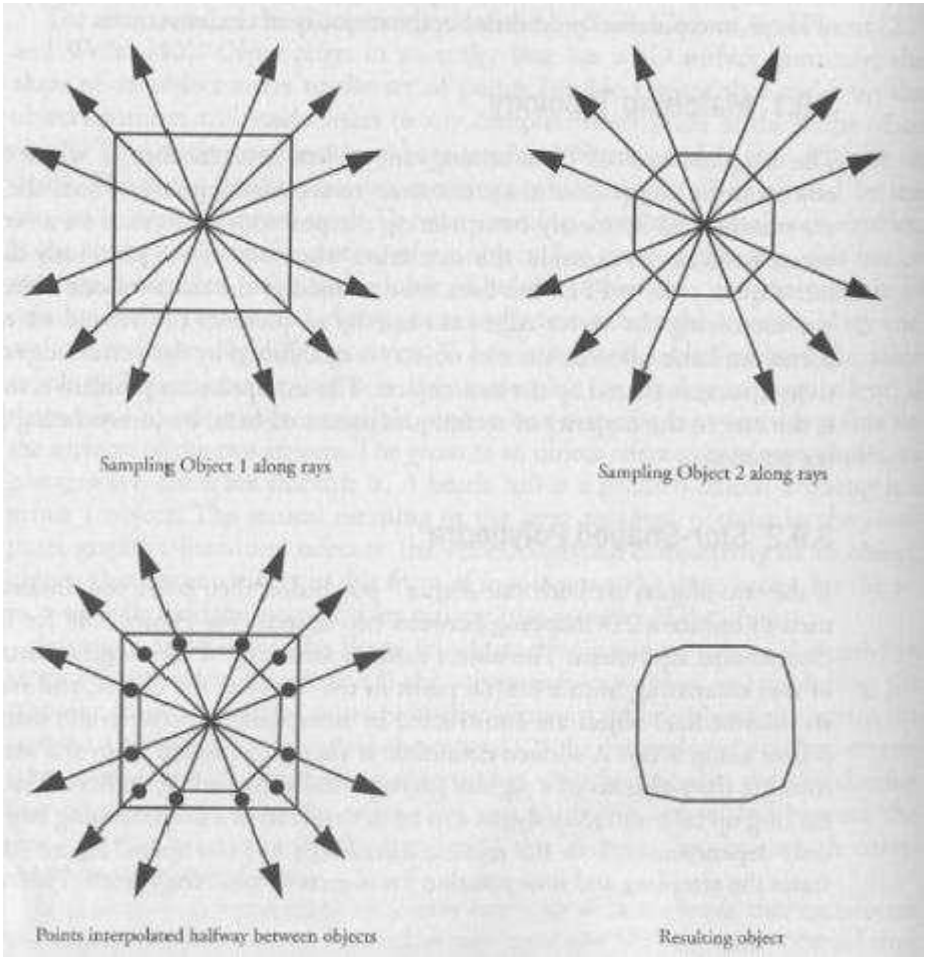


Abbildung 7.1: Morphing mit star shaped Polygonen (aus [Par02])

zusammengehörigen Punkten auf den beiden Polygonen, die die beiden Querschnitte am Ende des ersten und am Anfang des zweiten Polygones darstellen, bestimmen und diese dann für das Zwischenobjekt interpolieren. Ein einfaches Beispiel dazu sehen wir in Bild 7.1.

Für den Fall, dass die Polygone nicht star-shaped sind, bliebe zum Beispiel ein Coordinate Grid Approach, hier wird jeweils um die beiden Polygone oder Bilder ein Grid gelegt, welches zuerst interpoliert wird, um dann die neue Geometrie zu erzeugen. Wie das funktioniert, sehen wir in Bild 7.2.

Da in der Praxis die natürlichen Formen der Module aber wohl überwiegend zumindest eine grosse Ähnlichkeit mit star-shaped Objekten haben werden und man die benötigten zentralen Punkte am Anfang und Ende automatisch bestimmen kann (Anfang: Startpunkt mit epsilon in Ausrichtung des Modules, Ende: Der

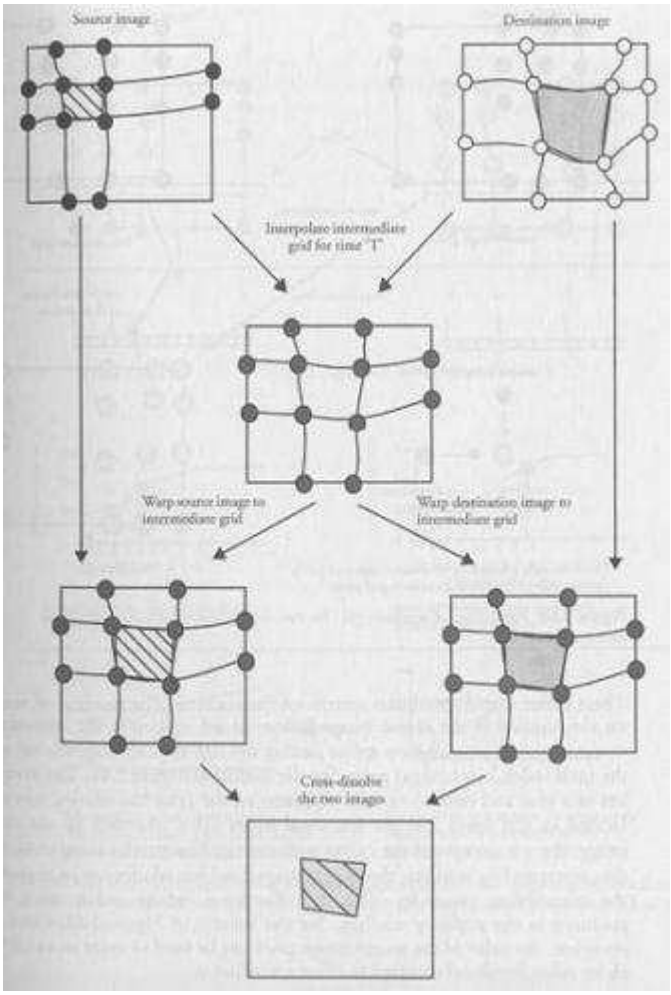


Abbildung 7.2: Coordinate Grid Approach (aus [Par02])

benutzte TOP-Anknüpfpunkt mit epsilon gegen die Normale des Anknüpfpunktes), erweist sich nach dem Bestimmen der Vertices ein Ansatz wie folgt trotzdem sicher als gut anwendbar:

1. Bestimmen der Polarkoordinaten für die Vertices der beiden Querschnitte im Verhältnis zu dem jeweiligen zentralen Punkt
2. Bestimmen von zusätzlichen Vertices in den Querschnitten durch Benutzen der Polarkoordinaten des jeweils anderen Querschnittes
3. Danach gewünschte Interpolation aller Vertex-Paare

Ein Beispiel mit zwei Querschnitten sehen wir in Bild 7.3.

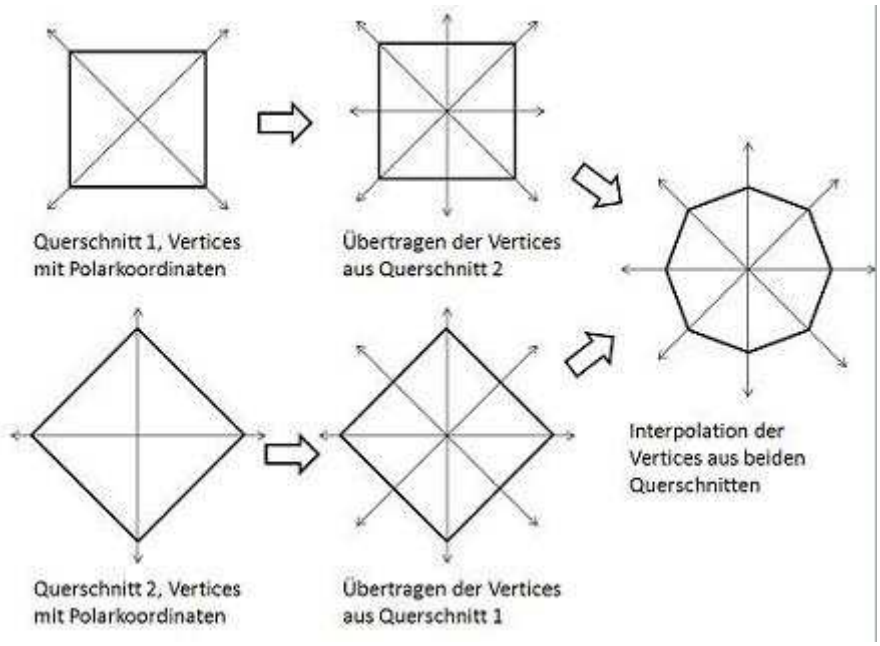


Abbildung 7.3: Verschmelzen von zwei star shaped Geometrien zur späteren Interpolation

Dies ist wirklich ein guter Ansatz, da die beiden gewählten zentralen Punkte später fast aufeinandergelegt werden (fast gilt nur wegen dem benutztem epsilon). Und wie oben schon geschrieben, eignet sich diese Vorgehensweise sicher auch für Querschnitte, die zwar nicht star-shaped sind, aber auch nicht zu sehr davon abweichen. Ein kleines Beispiel in dem einer der beiden Querschnitte nicht star shaped ist findet sich in Bild 7.4.

Ein ähnliches Verfahren würde sich auch für das Verschönern der Übergänge bei seitlich abzweigenden Modulen anbieten. Allerdings muss dann dort das Ganze etwas flexibler sein, weil mit Raycasting wie es oben beschrieben wurde, unter Umständen Vertices auf der falschen Seite des Modules gefunden werden. Dies verdeutlicht Bild 7.5, wo die beiden dicken Linien ein Modul und der dünnere Pfeil einen Ansatzpunkt mit seinem Normalenvektor darstellt. Das Raycasting würde hier in der Ebene, die durch die gestrichelte Linie dargestellt werden, durchgeführt werden und keinen für das Zwischenobjekt brauchbaren Querschnitt liefern.

In diesem Falle wäre es sicherlich besser, wenn um den Ansatzpunkt herum eine Kugel definiert wird (am besten mit dem Durchmesser des nachfolgenden Modules) und einige der Schnittpunkte zwischen Kugel und Vorgänger-Modul als Querschnitt für das Zwischenobjekt benutzt werden (wie genau das umgesetzt wird, wird etwas später noch erklärt).

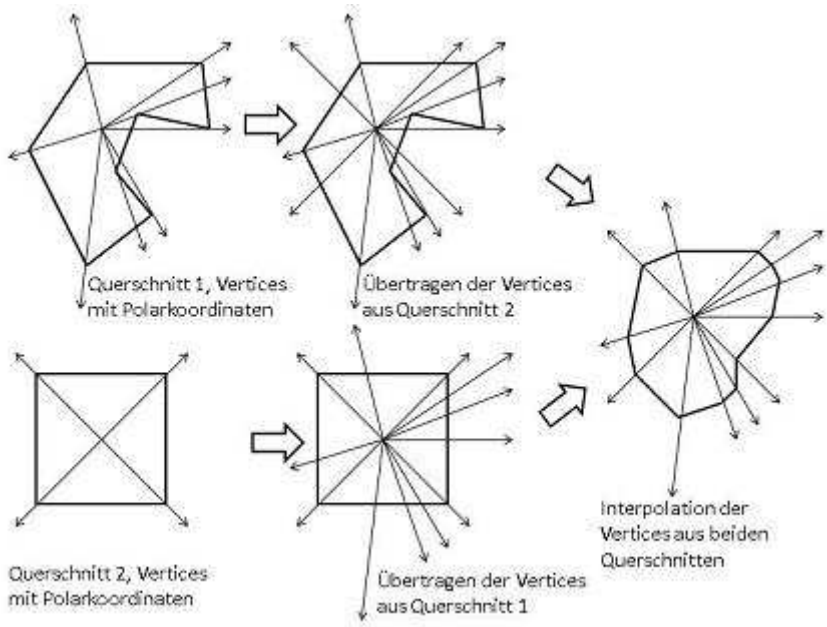


Abbildung 7.4: Verschmelzen von zwei Geometrien zur späteren Interpolation

Wie das Verschönern von seitlichen Übergängen dann aussehen kann, sehen wir in den Bildern 7.6, 7.7 und 7.8. In den ersten beiden Bildern sind jeweils zwei Module zu sehen, das gelbe Modul ist dabei ein Nachfolger des blauen Modules. Im ersten Bild sieht man, dass der Übergang in diesem Beispiel nicht sonderlich schön aussieht, der Anknüpfpunkt des blauen Modules liegt auf der Oberfläche bzw. Kante des Modules, die Basis des gelben Modules auf dem unteren Rand dieses Modules. Anstatt jetzt einen oder beide Punkte zu verlegen und eine gewisse Durchdringung der beiden Module in Kauf zu nehmen, kann nun ein Zwischenobjekt berechnet werden. Das Verlegen eines Anknüpfpunktes würde nämlich auch auf Kosten der eingestellten/gewünschten Länge der Module geht und im Einzelfall vielleich sogar dazu führen, dass ein nachfolgendes abzweigendes Modul auf der anderen Seite seines Vater-Modules wieder herausschauen kann. Im zweiten Bild ist dieses Zwischenobjekt in rot eingefügt und im dritten Bild sehen wir nur das Zwischenobjekt alleine.

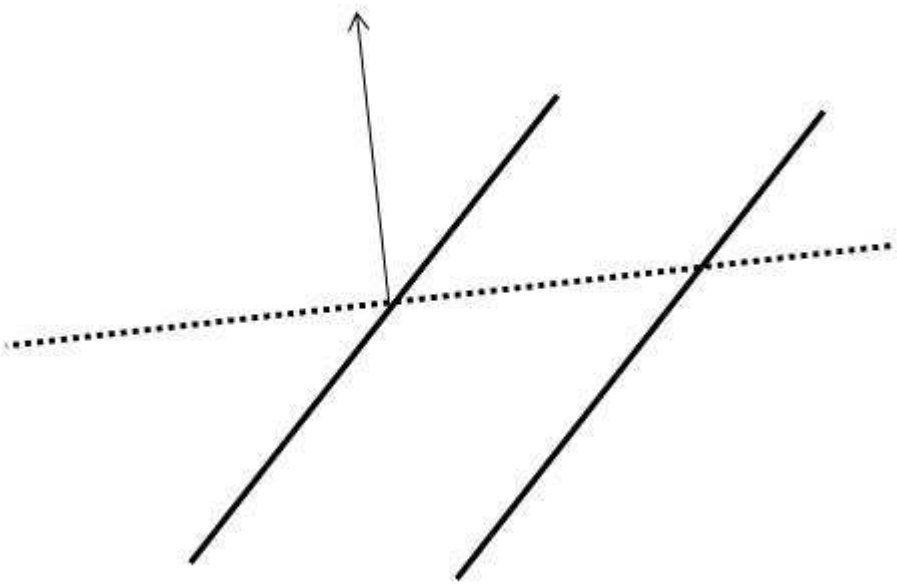


Abbildung 7.5: Ansatzpunkt, bei dem Raycasting nicht sinnvoll ist

Dieses Zwischenobjekt wurde wie folgt berechnet:

1. Bestimmen einer Menge ver1 von neuen Vertices am Anfang des gelben Modules mit Raycasting. Die Ebene des Raycastings liegt dabei in diesem konkreten Beispiel im lokalen Koordinaten-System des Modules bei einem y-Wert von 0,05 und ist paralell zur XZ-Ebene.

2. Bestimmen von weiteren Vertices als Menge ver2, diese werden bestimmt als Schnittpunkte des blauen Modules mit einer Kugel um den Anknüpfpunkt, von dem das gelbe Modul verzweigt.

3. Ordnen der beiden Vertex-Mengen zu einem Polygon-Zug und Bestimmen von Polarkoordinaten innerhalb der beiden Vertex-Mengen.

4. Einfügen von neuen Punkten in ver1 anhand der Polarkoordinaten der Vertices von ver2 und ebenso Einfügen von neuen Punkten in ver2 anhand der Polarkoordinaten von Vertices auf ver1.

5. Nun haben beide Mengen die gleiche Anzahl von Vertices, die jeweils paarweise zusammengehören, diese können nun nach Wunsch interpoliert und trianguliert werden.

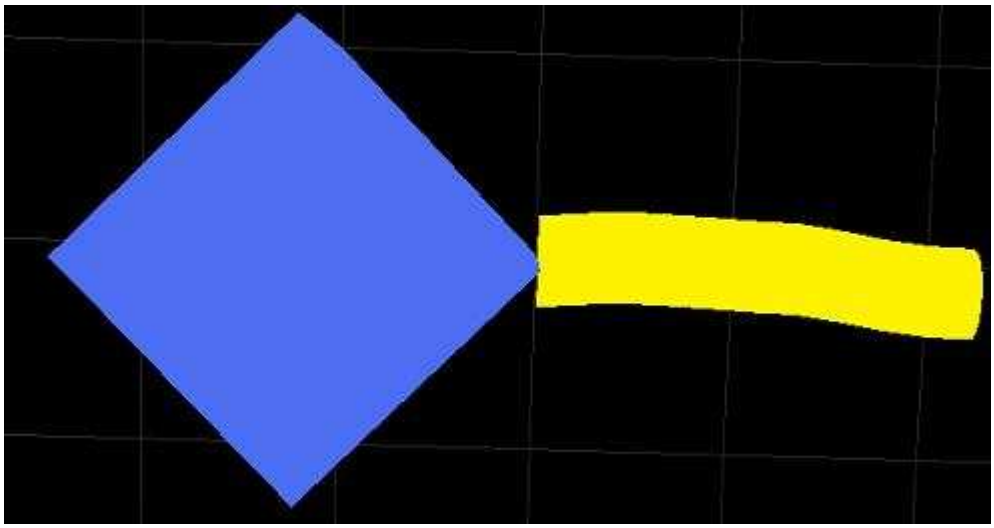


Abbildung 7.6: Der Übergang zwischen zwei Modulen ohne Zwischenobjekt

Für das Umsetzen dieser Vorgehensweise wurden zwei Näherungsverfahren eingesetzt, die auf der einen Seite das Berechnen vereinfachen und auf der anderen Seite noch dafür sorgen, dass das Zwischenobjekt nicht zu künstlich aussieht.

Ein Näherungsverfahren wird benutzt, um die Schnittpunkte zwischen Vorgänger-Modul und der Kugel zu bestimmen. Dazu wird jeder Vertex jedes Dreieck der Triangulierung auf die Entfernung zu dem Mittelpunkt der Kugel geprüft. Hat ein Vertex genau die gesuchte Entfernung, so wird dieser als ein Schnittpunkt gewählt. Anderenfalls gilt:

- Sind alle drei Vertices des Dreiecks näher an dem Mittelpunkt der Kugel als der Kugelradius, dann hat das Dreieck keinen Schnittpunkt mit der Kugel.
- Ist einer der Vertices näher und einer weiter weg, dann gibt es einen Schnittpunkt, es wird ein Punkt in der Mitte des Dreiecks bestimmt, mit dem das ursprüngliche Dreieck in drei kleinere zerteilt wird, die dann wieder betrachtet werden. Ab einem eingestellten Grenzwert für die Grösse des Dreiecks wird keine weitere Unterteilung durchgeführt und der Mittelpunkt einfach als Schnittpunkt zurückgegeben.
- Sind alle drei Vertices weiter weg von dem Mittelpunkt der Kugel als der Kugelradius es ist, dann wird das Dreieck nur weiter betrachtet (im Sinne der Unterteilung und rekursiven Weiterbetrachung wie im zweiten Punkt), wenn die Differenz zwischen echter Entfernung und Kugelradius eine Grenze unterschreitet, ansonsten wird das Dreieck ignoriert.

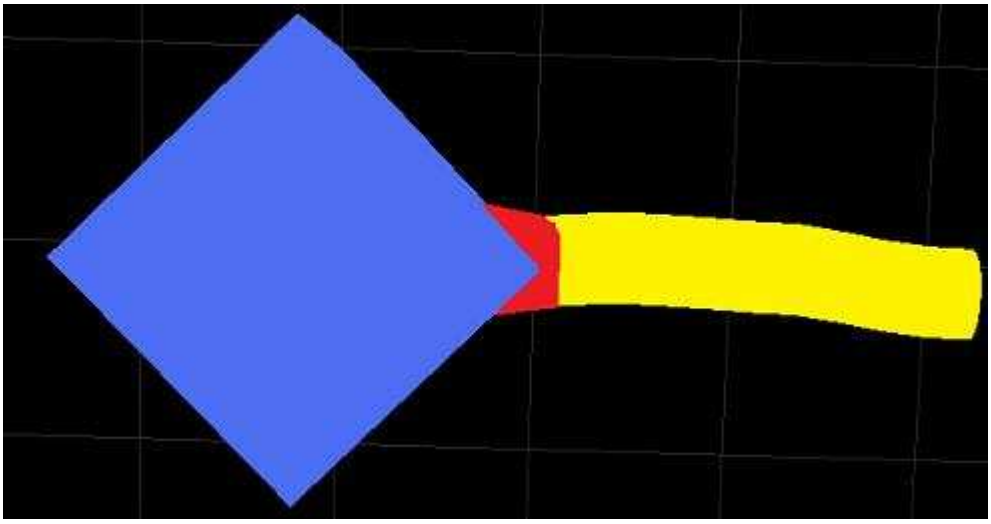


Abbildung 7.7: Der Übergang zwischen zwei Modulen mit Zwischenobjekt

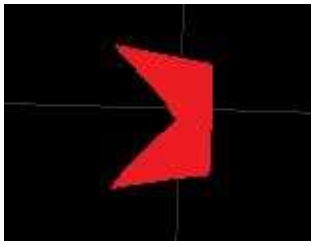


Abbildung 7.8: Nur das Zwischenobjekt

Ein zweites Näherungsverfahren wird für das Einfügen der neuen Vertices in die vorher geordneten star-shaped Polygone benutzt.

- Soll ein neuer Vertex eingefügt werden, dann wird zuerst die Polygon-Kante gesucht auf dem dieser Vertex liegen muss (das heisst, dass die Polarkoordinate des einen Vertex kleiner als die gewünschte Koordinate ist während die Polarkoordinate des anderen Vertex grösser ist).
- Diese Kante wird dann mit ihrem Mittelpunkt in zwei Teilkanten zerlegt.
- Je nach Polarkoordinate des Mittelpunktes im Vergleich zu der Polarkoordinate des einzufügenden Punktes wird dann in einer der beiden Teilkanten rekursiv weiter gesucht.
- Wird auch hier ein gewisser Grenzwert unterschritten, wird der Mittelpunkt der Kante als neuer Vertex eingefügt.

Abbildung 7.9: Ein natürlich aussehender Übergang zwischen Stamm und Ast (aus [Deu03]

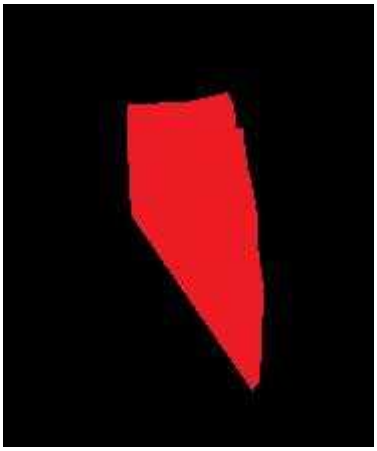


Abbildung 7.10: Zwischenobjekt mit einer Ellipse anstatt der Kugel

Übrigens: Die Anordnung der Vertices der beiden Querschnitte geschieht mit dem Algorithmus zur inkrementellen Bestimmung der konvexen Hülle einer Punktmenge. Dadurch kann es passieren, dass ein paar der vorher bestimmten Vertices (insbesondere bei dem Querschnitt mit der Kugel-Methode) wieder herausfallen, was dem Ergebnis aber nicht schadet.

Je nachdem wie man die Bestimmung der Punkte für ver2 umsetzt, ist sogar denkbar, dass damit ein ähnliches Ziel wie in Bild 7.9 erzielt werden kann. In Bild 7.10 wurden die Punkt von ver2 beispielsweise als Schnittpunkt des blauen Modules und einer Ellipse mit einem Mittelpunkt, der etwas tiefer als der Anknüpfpunkt liegt bestimmt. Würde man jetzt anstatt der verwendeten linearen Interpolation dieser Punkte noch ein besseres Verfahren nehmen, könnte man auf ein dem Bild 7.9 sehr ähnliches Ergebnis kommen.

Zusammenfassung dieses Abschnittes

Die folgenden Schritte werden zur Bestimmung der Anfangs-Breite eines Modules und zur Bestimmung von kleinen Zwischenobjekten zur Verschönerung der Übergänge durchgeführt:

1. Zuerst werden Vertices für die Querschnitte bestimmt, dies kann durch ein einfaches Raycasting-Verfahren oder durch die Bestimmung von Schnittpunkten des Modules mit einer Kugel, die um den gewünschten Punkt des Modules gelegt wird, geschehen.

2. Die Vertices können genutzt werden, um den Durchmesser der beiden Module an der Stelle zu berechnen. Für den Fall, dass das Nachfolger-Modul an einem TOP-Anknüpfpunkt des Vorgänger-Modules ansetzt, können diese Werte für eine Skalierung für das Nachfolger-Modul benutzt werden, die nicht auf dem in der Grammatik gegegeben Wert beruht.

3. Die Vertices der beiden Querschnitte werden jeweils mit dem Algorithmus zur inkrementellen Bestimmung der konvexen Hülle zu einem Polygonzug geordnet, dabei können eventuell ein paar nur näherungsweise bestimmte Punkte wieder herausfallen.

4. Nun werden zu den Vertices der beiden Querschnitte jeweils die Polarkoordinaten bestimmt.

5. Anhand der Polarkoordinaten werden nun in die beiden Querschnitte jeweils neue Vertices basierend auf den Vertices des jeweils anderen Querschnittes eingefügt.

6. Nun haben wir zwei Querschnitte mit der gleichen Anzahl Vertices und einer Zuordnung von jeweils zusammengehörigen Vertices. Diese Struktur kann zur Interpolation und Triangulierung des Zwischenobjektes benutzt werden.

Das beschriebene Verfahren eignet sich für jeden Anknüpfpunkt, an dem ein Modul in ein anderes Modul übergeht. Allerdings gibt es einige Faktoren, die das Ergebnis und das Aussehen des berechneten Zwischenobjektes beeinflussen.

Das wäre zum einen ein Epsilon als ein Teil Entfernung der neu zu bestimmenden Vertices von dem jeweiligen zentralen Punkt. Je grösser der Wert für Epsilon gewählt ist, desto grösser wird auch das berechnete Zwischenobjekt. Ein grösseres Zwischenobjekt bietet dabei bessere Möglichkeiten, um sehr unterschiedliche Geometrien der beteiligten Module ausgleichen zu können.

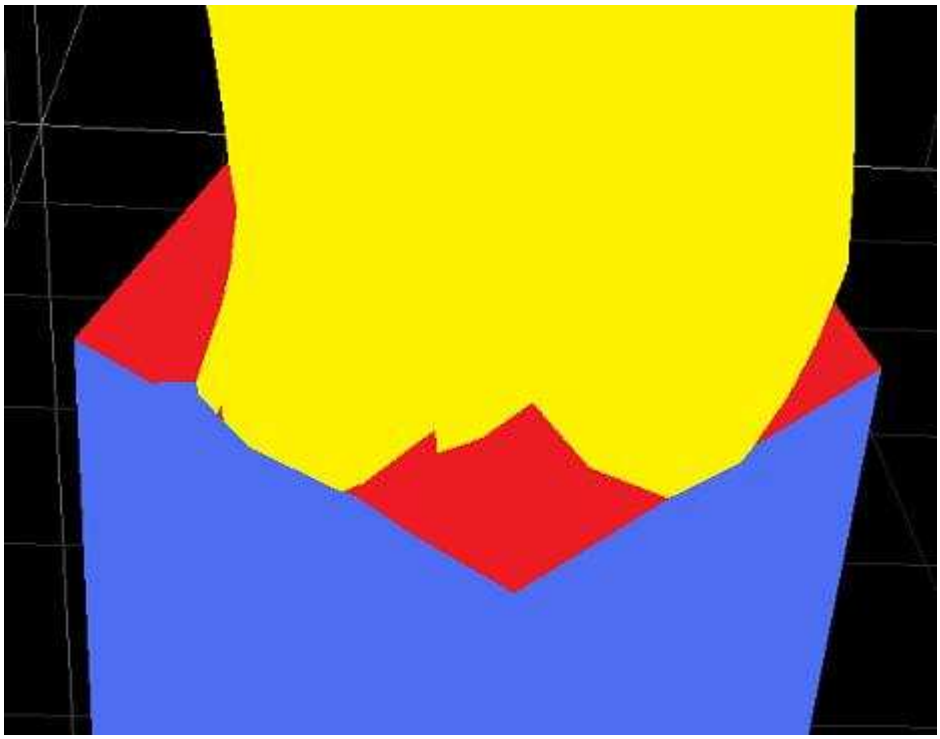


Abbildung 7.11: Problematischer Übergang an einem TOP-Anknüpfpunkt

Für den Fall, dass die Methode der Schnittbildung zwischen einer Kugel und dem Vorgänger-Modul zur Bestimmung der neuen Vertices eingesetzt wird, ist der Radius der benutzten Kugel ebenfalls einer der einflussnehmenden Faktoren. Ist der Radius grösser als der Durchmesser des nachfolgenden Modules, so kann ein ähnlicher Übergang wie in Bild 7.9 erzielt werden.

Die Anzahl der bei der Raycasting-Methode benutzten Strahlen bzw. der Abstand dieser Strahlen beeinflusst sehr stark das Aussehen des Zwischenobjektes. Je mehr Vertices bestimmt werden, desto glatter werden die Übergänge zwischen den Modulen und dem Zwischenobjekt sein.

Ausserdem werden zwei Grenzwerte für die Näherungsverfahren benutzt. Der erste für das Bestimmen der Schnittpunkte zwischen Kugel und den Dreiecken der Triangulierung und der zweite für das Einfügen von neuen Vertices anhand der Polarkoordinaten. Beide Parameter können die Berechnungen wesentlich beschleunigen und führen dazu, dass das Ergebnis nicht zu künstlich genau aussieht.

Bei wirklich ganz ungleicher Geometrie an einem TOP-Anknüpfpunkt kann dies mit einem grösserem Epsilon etwas besser ausgeglichen werden. Ein Beispiel dazu sehen wir in Bild 7.11, dort wird jeweils ein Quaderförmiges und ein Zylinderförmiges Modul verwendet. Mit einem grösserem Epsilon füllt das wieder in

rot dargestellte Zwischenobjekt etwas mehr von den Bruchstellen aus, da die für das Zwischenobjekt benutzten Vertices weiter oben bestimmt werden.

Insgesamt sehen wir aber auch, dass ein Ansatz mit gewissen Einschränkungen für die Geometrie hier von Vorteil wäre (zum Beispiel ähnlich wie die in Abschnitt 3.3 beschriebene Horn-Komponente der regelbasierten Objekterzeugung nach Deussen, bei denen jeweils mit runden Querschnitten gearbeitet wird). Aber dies gilt nur für die Top-Übergänge, bei den anderen seitlichen Übergängen muss man trotzdem flexibel sein.

7.3 Rekursion

Wie wir schon an dem Beispiel in 5.3 gesehen haben, kann Rekursion hilfreich sein. In dem Beispiel in 5.3 sollte der Stengel der Rose aus mehreren eigentlich gleichen Modulen hintereinander bestehen. Dies mit einer einzigen Regel, wie in Abbildung 7.12 angegeben ist, durchzuführen, würde zwei Probleme mit sich bringen.

Auf der einen Seite ist nicht klar, nach wievielen Ableitungen der Stengel am Ende ist. Eigentlich müsste er mit einer solchen Regel bis in das Unendliche wachsen. Ausserdem kann bei der Hintereinandersetzung auf diese Art jedes Modul des Stengels nur Seitentriebe des gleichen Types, also hier rosebranch, haben.

$$
\begin{aligned}
&rosestalk \rightarrow \\
&\quad < 1,1 > rosestalk[[< -3.0,-3.0 >, < 5.0,5.0 >, LIN]; \\
&\qquad < 135.0,135.0 >; TOP; 1; [0.0,0.25, LIN]; [0.0,1.0, LIN]]+ \\
&\quad < 3,3 > rosebranch[[0.0,75.0, LIN]; \\
&\qquad < 0.0,0.0 >; CENTER; 1; [0.0,1.0, LIN]; [0.0,1.0, LIN]]
\end{aligned}
$$

Abbildung 7.12: Naive Rekursion

Beide oben genannten Probleme waren im PlantAnimator lösbar, allerdings auf Kosten des Duplikation von schon vorhandenen Modulen. Indem man also das Modul rosestalk dupliziert (auch mehrfach) und mit einer Nummer versah, war es möglich, sowohl das Ende der Rekursion zu bestimmen als auch an verschiedenen Stellen des Stengels unterschiedliche Seitentriebe zu erhalten (in der Abbildung 7.13 die Module rosebranch1 und rosebranch2).

Das Ziel ist es, dass diese Art von Rekursion so in den PlantAnimator integriert wird, das sowohl das Hintereinandersetzen von gleichen Modulen mit definiertem Abbruch als auch eine möglichst flexible Steuerung der Seitentriebe erreichbar

$rosestalk1 \rightarrow$
$< 1,1 > rosestalk2[[< -3.0,-3.0 >,< 5.0,5.0 >, LIN];$
$< 135.0,135.0 >; TOP; 1; [0.0,0.25, LIN]; [0.0,1.0, LIN]]+$
$< 3,3 > rosebranch1[[0.0,75.0, LIN];$
$< 0.0,0.0 >; CENTER; 1; [0.0,1.0, LIN]; [0.0,1.0, LIN]]$

$rosestalk2 \rightarrow$
$< 1,1 > rosestalk3[[< -3.0,-3.0 >,< 5.0,5.0 >, LIN];$
$< 135.0,135.0 >; TOP; 1; [0.0,0.25, LIN]; [0.0,1.0, LIN]]+$
$< 3,3 > rosebranch2[[0.0,75.0, LIN];$
$< 0.0,0.0 >; CENTER; 1; [0.0,1.0, LIN]; [0.0,1.0, LIN]]$

Abbildung 7.13: Umsetzen der Rekursion im PlantAnimator auf Kosten der Modul-Definitionen

ist. Zu diesem Zweck reicht es aus, wenn in den Regeln der Grammatik auf der linken Seite jeweils eine Zahl im Sinne der Rekursionstiefe definiert wird. Dieser Parameter gibt dann an, in welcher Tiefe diese Regel anzuwenden ist. Der Scheduler erhöht die Tiefe bei jedem Übergang zwischen zwei Modulen des gleichen Types oder setzt sie auf 1, wenn es ein Übergang zwischen zwei unterschiedlichen Modulen oder der Verweis auf eine Teil-Grammatik ist. Der Scheduler beachtet die angegebene Tiefe bei dem Anwenden von Regeln. Dabei sollte es möglich sein, dass der Tiefen-Parameter auf der linken Regelseite als Intervall angegeben wird.

Möchte man dann eine Rekursion in der Pflanze haben, wo ein Stamm-Modul insgesamt fünfmal hintereinander gesetzt wird und wo alle Stammsegmente die gleichen Seitentriebe haben, dann braucht man lediglich zwei Regeln der folgenden Form:

Stamm[1-4] → Stamm + Seitentriebe_allgemein
Stamm[5] → Seitentriebe_nachWunsch + Krone

Dies könnte noch etwas vereinfacht und zu einer Regel zusammengefasst werden, würde aber auf Kosten der Les- und Überschaubarkeit der Grammatik gehen. Man müsste dazu ein Abgrenzungs-Symbol einführen, um kennzeichnen zu können, was alles während und was nach der Rekursion an Kindern erzeugt wird. Möchte man zusätzlich, dass jedes Modul des Stammes unterschiedliche Seitentriebe haben kann, dann können trotzdem fünf Regeln verwendt werden, diese basieren trotzdem nur auf einem einzigen Modul für den Stamm.

Stamm[1] → Stamm + Seitentrieb1
Stamm[2] → Stamm + Seitentrieb2

Stamm[3] → Stamm + Seitentrieb3
Stamm[4] → Stamm + Seitentrieb4
Stamm[5] → Krone

7.4 Wiederverwendbarkeit

Durch das verbesserte Modellieren und Kontrollieren der Zeitachse sind schon einige grundlegende Parameter, deren Verankerung in den Modulen die Wiederverwendbarkeit negativ beeinflusst haben, nun über die Grammatik veränderbar. Ausserdem ist durch das Einführen der möglichen Rekursion eine weitere Quelle für unnötige Modul-Duplikationen entfernt worden. Nun muss nun kein Modul mehr dupliziert werden, um einen längeren Stamm aus gleichen Modulen zu modellieren.

Weitergehend blieben nur noch die Parameter für Texture- und Farb-Informationen, die in den Modulen gespeichert werden und die die Wiederverwendbarkeit negativ beeinflussen. Normale Farb-Informationen werden normalerweise mit einem Farb-Wert pro Vertex der zugrundeliegenden Geometrie beschrieben, daher ist es keine praktikable Lösung, diese Informationen in die Grammatik zu verlagern. Das Verlagern der Texture-Informationen in die Grammatik hingegen sollte kein Problem sein. Allerdings wäre es dann wünschenswert, wenn der Benutzer aus der Grammatik nicht auf eine beliebige Datei auf seiner Festplatte verweisen kann. Dies könnte bei dem Austausch von Grammatiken und modellierten Pflanzen Probleme erzeugen und würde bei Angabe eines kompletten Dateipfades die Grammatik wieder etwas unübersichlicher machen. Hier wäre wohl eine Vorgehensweise wie bei den Funktionen vorzuschlagen. Das bedeutet, dass die Texturen von der Grammatik getrennt erfasst und dann nur durch ein kurzes Alias in der Grammatik verwendet werden.

7.5 Umwelt-getriebene Animation

In diesem Abschnitt wollen wir uns einmal Gedanken darüber machen, welchen Einfluss das Umsetzen der umwelt-getriebenen Animation auf den PlantAnimator hat. Dabei wurde in Kapitel 6 im Wesentlichen schon beschrieben, wie die umwelt-getriebene Animation so umgesetzt werden kann, dass sie mit dem bisherigen Konzept des PlantAnimators zusammenarbeitet. Offen gelassen wurde das Bestimmen der tatsächlichen Stärke und die Untersuchung, welchen Einfluss das Umsetzen der umwelt-getriebenen Animation auf die Grammatik des PlantAnimators hat.

Erinnern wir uns noch einmal, dass die Gesamt-Stärke s für den Einfluss, den der Wind auf ein Modul nimmt, mit folgender Formel bestimmt wird:

$s = pwl * wb$

dabei ist

pwl Projezierte Windlänge: Wird noch mit einer gewünschten Maximalstärke für den Wind auf das Intervall 0 bis 1 skaliert.

wb Wind-Beeinflussbarkeit: Berechnet sich aus Festigkeit des Modules in Verbindung mit Länge und Breite.

Das tatsächliche Bestimmen der Wind-Beeinflussbarkeit war dabei in dem Kapitel 6 noch offen gelassen worden, es soll nun hier etwas näher betrachtet werden. Dazu führen wir noch einmal die Faktoren vor Augen, die Einfluss auf die Wind-Beeinflussbarkeit nehmen sollten und überlegen uns dann, wie der Einfluss dieser Faktoren aussieht und woher die Werte für diese Faktoren kommen.

- Festigkeit f des Materiales
- Länge l des Modules
- Der Durchmesser d des Modules

Dabei ist die Festigkeit des Materiales ein Wert, der durch den Benutzer vorgegeben werden muss und der in der Grammatik stehen sollte. Je fester und damit widerstandsfähiger ein Material ist, desto weniger wird es sich durch Wind oder andere äussere Faktoren beeinflussen lassen. Die Länge des Modules fliesst über die zugrundeliegende Scherung schon in das Ergebnis ein. Diese sorgt dafür, dass die Abweichung linear in Bezug auf die Länge erfolgt, d. h. dass das Ende eines Modules doppelt so weit verschoben wird wie die Mitte. Der Durchmesser eines Modules kann durch ein einfaches Verfahren wie in Abschnitt 7.2 vorgestellt bestimmt werden. Der konkrete Durchmesser ergibt sich dann durch das Anwenden der für ein konkret in der Pflanze benutztes Modul vorgesehenen Skalierung auf den im Modul-Editor berechneten Durchmesser.

Es reicht also aus, wenn die beiden Faktoren f und d zu der Wind-Beeinflussbarkeit zusammengesetzt werden. Da sich eine höhere Festigkeit und ein grösserer Durchmesser beides negativ auf die Wind-Beeinflussbarkeit auswirkt (ein Ausnahmefall wäre der Durchmesser, wenn das Modul eine sehr unregelmässige Geometrie hat), wird die folgende Formel vorgeschlagen:

$wb = \frac{1}{f*d}$

Die damit berechnete Wind-Beeinflussbarkeit sollte allerdings noch skaliert werden, insbesondere damit sie bei kleinem f und d nicht zu gross wird und dann zu unglaubwürdigen Ergebnissen führen kann. Um das gleiche Ziel zu erreichen, würde es auch ausreichen, wenn als kleinster Wert für f und d nur 1 erlaubt wird. Für Die Festigkeit f ist das sicher keine Einschränkung, weil die benutzte Skala dann einfach so festgelegt wird, dass 1 der minimale Wert auf dieser Skala ist. Für den Durchmesser d könnte es sein, dass 1 nicht der minimal mögliche Wert ist, aber dann bliebe noch die Möglichkeit, die Skala derart zu verschieben, dass 1 doch der minimale Wert ist. Aufgrund der Vergleichbarkeit der Werte vorher und nachher sollte dies aber keine additive Verschiebung sondern eine multiplikative Verschiebung sein. Ist beispielsweise der minimale Wert 0.5 vor der Verschiebung, dann sollten alle Werte mit 2 multipliziert werden, damit aus einem vorher-Wert 1 dann der Wert 2 wird und die vorher bestehende Relation zwischen 0.5 und 1 erhalten bleibt.

Neben diesen Berechnungen hat sich ebenfalls in Kapitel 6 gezeigt, dass es bei der Umsetzung der umwelt-getriebenen Animation verschiedene Optimierungsmöglichkeiten für die Effizienz des Verfahrens gibt und dass daher ein weiterer Wert in der Grammatik angeben soll, wieviel der Pflanze durch das Berechnen der umwelt-getriebenen Animation beeinflusst werden soll und demzufolge auch, wieviel der Pflanze durch etwas mehr zufälligeres Bewegen beeinflusst werden soll.

7.6 Die neue Grammatik

Nachdem in den vorherigen Abschnitten einiges zusammengetragen und betrachtet wurde, was Einfluss auf die Grammatik hat, wollen wir uns einmal eine dementsprechend erweiterte Grammatik anschauen:

$M_j[x] \rightarrow$ <m,k>M_i{b,e,l,v,f,t}$[[a_1, a_2, F_a, b_a, e_a]$;<$b_1$,$b_2$>;Ap;Rej;$[w_1, w_2, F_w, w_a, w_a]$; $[h_1, h_2, F_h, b_h, e_h]]$ + <n,j> M_v{b,e,l,v,f}[...]

Allgemeine Konstrukte:
<x,y> : Zufallszahl zwischen x und y
$[x, y, z, b, e]$: Beschreibt jeweils eine Parameteränderung mit Startwert x, Endwert y und Funktionsalias z. Sowie einem prozentualen Wert b, der angibt, wielange der Parameter zu Beginn der Modul-Entwicklung auf dem Wert x bleibt, und dem prozentualen Wert e, der angibt, wielange der Parameter am Ende der Modul-Entwicklung auf dem Wert y stehen bleibt.

Konkrete Elemente der oberen Beispiels-Regel:

<m,k>: Zufallszahl für die Anzahl der erzeugten Pflanzenelemente

{b,e,l,v,f,t}: Angabe eines Intervalles für die Entstehung des Modules M_i als Kind von M_j, die Angabe der Werte für b und e erfolgt prozentual. l gibt die Lebenszeit des Modules M_i an, v steuert das Verhalten von M_i am Ende der Lebenszeit, f ist die Festigkeit des Modules und t die verwendete Texture.

$[a_1, a_2, F_a, b_a, e_a]$: Winkel zwischen zwei Pflanzenelementen

$[w_1, w_2, F_w, b_w, e_w]$: Breite dieses Pflanzenelementes

$[h_1, h_2, F_h, b_h, e_h]$: Länge dieses Pflanzenelementes

$<b_1, b_2>$: Drehwinkel in Bezug auf das Vorgängerelement M_j

Ap: Anknüpfpunkt (BOTTOM, TOP, CENTER, ALL, Benutzung von NOT erlaubt)

Rej: Verjüngungsfaktor: Um wieviel soll sich die Breite am Ende eines Pflanzenelementes von der Anfangsbreite unterscheiden

Der durch eckige Klammern auf der linken Regelseite eingeschlossene Wert x steht für die Rekursionstiefe wie in Abschnitt 7.3 besprochen. Neu ist die Erweiterung der Parameterangabe um zwei weitere Werte für die Verzögerung vor und nach der eigentlichen Parameterentwicklung. Ausserdem ist noch die Angabe von 6 Werten dazugekommen, die neben dem Zeitpunkt des Entstehens des entsprechenden Modules auch seine Lebenszeit und das Verhalten nach der Lebenszeit angeben. In diesen 6 Werten steckt ebenfalls die Information über die zu verwendende Texture als auch die Festigkeit des Modules.

Zusätzlich muss im Grammatik-Editor ein Wert für die Umsetzung der umwelt-getriebenen Animation angegeben werden. Dieser gibt an wieviel der Bewegungen der Pflanze im Rahmen der umwelt-getriebenen Animation basierend auf dem Wind berechnet wird und wieviel von der Bewegung zufällig ist. Der Wert wird von dem Scheduler ausgelesen aber nur bei den Grammatiken benutzt die in der Szene als vollständige Pflanze dargestellt wird. Wird eine Grammatik nur als Teil einer grösseren Pflanze benutzt und damit nicht direkt dem Scheduler auf oberster Ebene übergeben, dann wird der Wert dort ignoriert.

Kapitel 8

Virtuelle Welten und Effizienz-Steigerungs-Möglichkeiten für die umwelt-getriebene Animation

Nachdem wir in den vorhergehenden Kapiteln die Möglichkeiten des PlantAnimators für einzelne Pflanzen kennengelernt haben, wollen wir jetzt noch einen besonders wichtigen Punkt untersuchen.

In diesem Kapitel wird es um die Effizienz der Animation im Rahmen einer virtuellen Welt gehen und insbesondere die Effizienz der umwelt-getriebene Animation, die in Kapitel 6 schon angesprochen wurde.

Dabei wollen wir untersuchen, welche Optimierungsmöglichkeiten es in virtuellen Welten im Allgemeinen und durch das Grundkonzept des PlantAnimators im Besonderen gibt. Die grössten Optimierungsmöglichkeiten drehen sich dabei um das Blickfeld und die Entfernung zwischen dem Betrachter und den betrachteten Objekten.

Durch das Ausnutzen des Blickfeldes ist es unter Umständen möglich, dass Pflanzen erkannt werden können, die nicht animiert werden müssen und damit keine Rechenzeit verbrauchen. Das bedeutet aber, dass die Pflanzen möglichst effizient aktualisiert werden müssen, wenn diese wieder in den sichtbaren Bereich kommen. Dieser Ansatz mit dem Blickfeld bringt allerdings nur dann eine Verbesserung, wenn die Ersparnis an Rechenzeit durch die Nicht-Animation der Pflanzen grösser ist als die zu erwartenden Kosten. Um die Unterscheidung in aktive und inaktive Pflanzen vornehmen zu können, muss nämlich fortwährend die Berechnung des Blickfeldes des Betrachters erfolgen. Das Ziel dieses Kapitels wird es sein, aufzuzeigen, dass die Unterscheidung in aktive und inaktive Pflanzen sinnvoll ist. Dies liegt einerseits daran, dass die Berechnung des Blickfeldes sehr effizient möglich

ist, andererseits aber auch daran, dass der PlantAnimator sehr effizient arbeitet. Aus letzterem Grund kostet das „Aktivieren“ einer inaktiven Pflanze nur wenig mehr als ein normaler Animationsschritt.

Das Ausnutzen der Enfernung für die (zumeist statische) Darstellung von Objekten in virtuellen Welten wird schon häufig genutzt. Wir wollen hier zeigen, dass der PlantAnimator ein ähnliches Konzept für das Berechnen und Darstellen der Animation bietet, welches durch den Modellierer einer Szene und von Pflanzen einfach (das heisst insbesondere ohne Kenntnisse der zugrundeliegenden Prozesse oder Programmierkenntnissen) und intuitiv eingesetzt werden kann .

8.1 Virtuelle Welten

Zuerst einmal muss erwähnt werden, dass es bei den hier beschriebenen virtuellen Welten nicht in erster Linie um Konzepte der sogenannten virtuellen Realität geht. Es werden also insbsondere auch keine speziellen Ein- und Ausgabegeräte verwendet (wie zum Beispiel 3D-Brillen, sogenannte Head Mounted Displays), was aber nicht bedeuten muss, dass es nicht möglich wäre, die hier definierten virtuellen Welten auch auf diese Weise darzustellen. Eine virtuelle Welt wird hier mehr als das verstanden, was auch in verschiedenen Computerspielen und Filmen schon eingesetzt wird. Es wird also versucht, die reale Umgebung virtuell nachzubilden, und das Ziel ist, dass die virtuelle Welt als möglichst glaubwürdig empfunden wird, während man in dieser virtuellen Welt Effekte umsetzen kann, die in der Realität entweder gar nicht möglich oder nur mit hohen Kosten verbunden wären.

Durch Fortschritte im Rahmen der Computer-Entwicklung (insbesondere der Preisentwicklung bei Computer-Technologien wie Hauptspeicher) rücken nun auch bei virtuellen Welten verstärkt Aspekte in den Vordergrund, die bisher ignoriert oder nur sehr oberflächlich behandelt wurden. Trotzdem ist die Entwicklung von virtuellen Welten immer noch sehr stark von dem Effizienz-Gedanken geprägt, die Effizienz von Verfahren ist zumeist wichtiger als die botanisch oder physikalisch korrekte Umsetzung. Daher muss jedes Verfahren, welches in virtuellen Welten umgesetzt oder eingesetzt werden soll, sehr effizient sein.

Zu dem Erstellen einer virtuellen Welt gehören Teilaufgaben wie das Modellieren von Terrain, das Modellieren der Wasserverteilung aufbauend auf dem Terrain und das Modellieren von Pflanzenbewuchts aufbauend auf der Wasserverteilung und dem Terrain. Hierbei beeinflussen sich Terrain und Wasserverteilung gegenseitig, das Terrain beeinflusst das fliessende Wasser, welches wiederum für Erosion sorgt. Das alles sind Punkte, die in dieser Arbeit nicht weiter verfolgt werden, interessierte Leser werden an [Kir07] verwiesen.

In dieser Arbeit gehen wir bei der virtuellen Welt davon aus, dass diese schon fertig definiert und modelliert ist und wir betrachten, welche Aspekte einer virtuellen Welt zur Verbesserung der Effizienz des gesamten Pflanzen-Bewuchses und damit der ganzen Welt benutzt werden können. Ausnutzen wollen wir dabei Entfernungs- und Sichtbarkeits-Informationen, um die Effizienz der umweltgetriebenen Animation für die Welt insgesamt verbessern zu können. Für die Information, welche Teile der Welt sichtbar sind, muss das Blickfeld berechnet werden, dieses wird auch Viewing Frustum genannt. Ausserdem ist es sehr sinnvoll, die Welt mit einem Raumaufteilungs-Verfahren so aufzuteilen, dass Information zu Entfernung und Lage von Objekten (und damit auch Pflanzen) nicht auf der Basis von jedem einzelnen Objekt behandelt werden müssen.

8.1.1 Raumaufteilung

Die Raumaufteilung einer virtuellen Welt kann auf verschiedene Arten geschehen. Bewährt haben sich entweder eine AABB-Hierarchie (axis aligned bounding boxes) oder OBB (oriented bounding box), also Bounding-Boxen für die Objekte, die entweder an den Achsen ausgerichtet werden (AABB) oder frei im Raum stehen (OBB).

Eine weitere Alternative wären eine Beschreibung der Bounding Boxes anhand von k-DOP oder k-Oriented Discrete Polytopes, also die Benutzung von Polyedern, die Objekte noch besser umschliessen. Dies werden wir hier aber nicht weiter betrachten, da die gewünschte Raumaufteilung für die virtuelle Welt nicht herunter bis zu jedem einzelnen Objekt gehen wird und insbesondere das Berechnen der groben Sichtbarkeit in diesem Falle ausreicht. Grob bedeutet, dass einige Pflanzen zuviel als sichtbar eingestuft werden dürfen, als eigentlich sichtbar wäre, wenn dafür die Berechnungen des Blickfeldes effizient zu realisieren sind.

Ein Unterschied zwischen AABB und OBB liegt in der (hier nicht benötigten) Kollisionserkennung, dort ist AABB auch wegen der schlechteren Umschliessung der Objekte um einiges langsamer als OBB, benötigt dafür aber weniger Speicherplatz für die Boxen.

Für unsere virtuelle Welt wollen wir nun eine AABB-Hierarchie benutzen, um von den Positionen der einzelnen Objekte und Pflanzen abstrahieren zu können. Die Aufteilung kann einmal nach dem Erstellen der Pflanzenverteilung bestimmt und muss dann nur noch aktualisiert werden. Dabei ist darauf zu achten, dass sich weder zuviele noch zuwenige Pflanzen in einer einzelnen Box auf der untersten Ebene der Hierarchie befinden. Ausserdem sollten die Boxen auf der untersten Ebene weder zu klein noch zu gross werden.

Die erste Bedingung legt eine variable Hierarchie nahe, bei der man Grenzwerte angibt, bei welchen eine Box entweder rekursiv auf kleinere Boxen aufgeteilt oder eine Box mit entsprechenden Nachbarn wird zu einer grösseren Box zusammengefasst, während die zweite Bedingung eher für eine feste Hierarchie nahelegt. In der Praxis wird wahrscheinlich eine Mischung aus beiden eine gute Wahl sein.

Offen bleibt zu dem Thema noch die Fragestellung, wie mit Pflanzen, die genau auf dem Rande zwischen zwei Boxen wachsen, umzugehen ist. Diese könnten in beide Boxen gesetzt werden, wenn sichergestellt ist, dass für die gleiche Pflanze dann nicht zweimal in einem Frame die gleichen Berechnungen durchgeführt werden. Oder aber, wenn die Kosten für das doppelte Berechnen sehr gering (beispielsweise bei sehr kleinen Pflanzen) sind. Genauer werden wir das in dem noch folgendem Abschnitt zur Modellierung eines ganzen Ökosystemes durch eine geeignete Raumaufteilung sehen. Dort werden in unterschiedlichen Schichten unterschiedlich grosse Boxen für unterschiedlich grosse Pflanzen benutzt, grosse Pflanzen (für die die doppelte Berechnung recht teuer wären) finden sich dort in grossen Boxen, was bedeutet, dass es insgesamt weniger Randbereiche zwischen diesen Boxen gibt, als wenn die Szene in viele kleinere Boxen eingeteilt sind.

Aufgrund des Querbezuges zwischen der Raumaufteilung und der Modellierung eines Ökosystemes werden wir die Überlegungen zu einer zweckmässigen Raumaufteilung für virtuelle Welten ab Seite 145 fortsetzen.

8.1.2 Blickfeld

Ein wichtiger Punkt für die folgenden Optimierungen und in virtuellen Welten allgemein ist das Berechnen des Blickfeldes (auch View Frustum genannt). Typischerweise wird das View Frustum als rechteckige Pyramide, die von einem Augpunkt ausgeht, dargestellt (siehe auch das Bild 8.1).

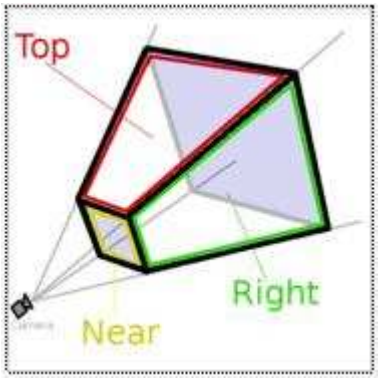


Abbildung 8.1: Darstellung des View Frustum (aus [Wik07]

Das View Frustum ergibt sich durch insgesamt sechs begrenzende Ebenen, neben den gezeigten near, top und right noch die Ebenen far, bottom und left. Objekte die näher als die Ebene near oder weiter weg als die Ebene far von der Kamera sind, müssen nicht weiter betrachtet werden. Aufbauend auf dieser Definition wird dann das sogenannte View Frustum Culling definiert. Dieses beschreibt den Prozess des Bestimmens von Objekten, die ausserhalb des Blickfeldes liegen.

Das View Frustum Culling (VFC) mit Bounding Boxes kann grundsätzlich auf zwei verschiedene Arten erfolgen. Die erste Variante sehen wir in Bild 8.2. Dort wird jede Box a anhand der Perspektive des Blickfeldes in eine neue Box b transformiert, dann wird eine neue minimale Box c bestimmt. Danach kann der Test, ob die jeweilige Box sichtbar ist, dann mit zwei AABB Objekten durchgeführt werden, was sehr einfach und mit nur sechs Koordinaten-Vergleichen möglich ist.

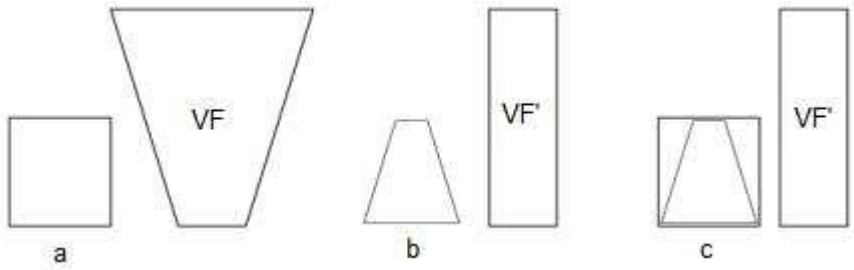


Abbildung 8.2: VFC mit perspektivischer Transformation (aus [AM00])

Der zweite Ansatz entspricht dem Vergleich der Boxen mit allen Ebenen des View Frustum (VF). Die Box wird dabei zu jeder Ebene in Beziehung gesetzt und das Ergebnis dieses Tests lautet entweder OUTSIDE, INSIDE oder INTERSECTING. Ist eine Box INSIDE von allen Ebene des VF, dann ist das Gesam-

tergebnis des Tests INSIDE und es muss keine rekursive Betrachtung der in der Box möglicherweise enthaltenen kleineren Boxen durchgeführt werden. Ist eine Box Outside von einer Ebene des VF, dann ist das Gesamtergebnis OUTSIDE und auch hier muss keine rekursive Weiterbetrachtung von kleineren Boxen erfolgen. In allen anderen Fällen ist das Gesamtergebnis INTERSECTING, was nichts anderes bedeutet, dass keine klare Entscheidung für diese Box gefällt werden konnte. Existiert noch eine rekursive Unterteilung dieser Box, dann werden alle kleineren Boxen mit dem VF getestet, anderenfalls muss man diese Box als sichtbar zählen, auch wenn wahrscheinlich nicht alle Objekte innerhalb dieser Box sichbar sind. Dies sorgt insbesondere bei grossen Boxen dafür, dass zuviele Pflanzen als sichtbar eingestuft und animiert sowie gezeichnet werden, aber wir werden bei der später konkret vorgeschlagenen Raumaufteilung sehen, dass dies kein zu grosser Nachteil sein muss.

Das Verfahren insgesamt lässt sich durch einfachen Pseudo-Code beschreiben, der positive Halbraum ist dabei derjenige Halbraum einer Ebene e in den der Normalenvektor von e zeigt, bei diesem Pseudo-Code wäre also darauf zu achten, dass die Normalenvektoren der Ebenen des VF alle nach aussen zeigen. Als Ergebnis bekommen wir eine Menge von sichtbaren Boxen innerhalb der Welt (wie diese Menge gespeichert wird, lassen wir mal offen).

```
 1 int OUTSIDE = -1;
 2 int INSIDE = 1;
 3 int INTERSECTING = 0;
 4
 5 int testBoxMitEbene(Box b, Ebene e){
 6         boolean pos=false, neg=false;
 7         for jeden Vertex v von b do
 8                 if v befindet sich im positiven Halbraum von e
 9                         pos = true;
10                 else
11                         neg=true;
12         if neg == false
13                 return OUTSIDE;
14         if pos == false
15                 return INSIDE;
16         return INTERSECTING;
17 }
18
```

```
19 int testBoxMitVF(Box b, VF vf){
20         boolean out = false, ins = false, inter = false;
21         for jede Ebene e von vf do
22                 int ret = testBoxMitEbene(b,e);
23                 if ret == OUTSIDE
24                         out = true;
25                 else if ret == INSIDE
26                         in = true;
27                 else if ret == INTERSECTING {
28                         inter = true;
29                         break;
30                 }
31         if inter == true
32                 return INTERSECTING;
33         if out == false
34                 return INSIDE;
35         if ins == false
36                 return OUTSIDE;
37         return INTERSECTING;
38 }
39
40 void testBoxen(Box b){
41         int ret = testBoxMitVF(b,vf);
42         if ret == INSIDE
43                 Füge b und alle Kinder zu den sichtbaren Boxen hinzu;
44         if ret == OUTSIDE
45                 return;
46         if weitere Unterteilung möglich {
47                 testBoxen(b.getFirstChild());
48                 testBoxen(b.getSecondChild());
49                 testBoxen(b.getThirdChild());
50                 testBoxen(b.getFourthChild());
51         }
52 }
```

In [AM00] findet sich eine Vielzahl von Optimierungsmöglichkeiten für das VFC basierend auf dem zweiten Ansatz, diese beinhalten:

1. Anstatt alle Vertices einer Box mit den Ebenen des VF auf die Lage selbiger zueinander zu testen, werden nur zwei Vertices betrachtet, diese werden als n-Vertex (negativ far point) und p-Vertex (positiv far point) bezeichnet und sind die beiden Vertices der Diagonale durch die Box, deren Ausrichtung

dem Normalenvektor der Ebene am nähesten liegt. Der n-Vertex ist dann derjenige von beiden, der den kleineren Abstand zu der Ebene von den beiden Punkten hat. Ist der n-Vertex ausserhalb der Ebene, dann ist auch die Box ausserhalb der Ebene. Liegt hingegen der p-Vertex innerhalb der Ebene, so ist auch die Box innerhalb der Ebene. Anderenfalls schneidet die Box die Ebene.

2. Wenn eine Box im letzten Schritt als OUTSIDE erkannt wurde, dann speichert man nach Möglichkeit ab, welche Ebene für das Ergebnis verantwortlich war und fängt bei der nächsten Überprüfung mit dieser Ebene an.

3. Durch das Einteilen des VF in acht Oktanten (siehe auch Bild 8.3 (b))) kann eine Verbesserung der Tests für Boxen, deren Zentrum innerhalb des VF liegt, erreicht werden. Ursprünglich wurde dieser Test für Bounding Spheres eingeführt, wenn die Kugel innerhalb der drei äusseren Ebenen des Oktanten, in dem der Mittelpunkt der Kugel liegt, ist, dann ist sie auch innerhalb des kompletten VF. Dieses Vorgehen kann auf Boxen erweitert werden, funktioniert aber nur mit Boxen, bei denen der grösste Abstand zwischen dem Zentrum der Box und einer Ecke der Box kleiner ist als der kleinste Abstand zwischen dem Zenrum der VF und einer Ebene des VF.

4. Wenn bei dem Test einer größeren Box festgestellt wird, dass diese komplett INSIDE einer der einzelnen VF-Ebenen ist, dann gilt dies auch für alle Kinder dieser Box. Diese Ebene muss also bei rekursiven Schritten nicht mehr geprüft werden.

5. Man kann sich die Benutzer-Bewegungen in der Welt anschauen. Steht dieser still, so braucht kein erneutes VFC durchgeführt zu werden. Hat er sich nur etwas nach rechts gedreht, so können im allgemeinen die Boxen, die vorher aufgrund der linken Ebene des VF als OUTSIDE deklariert wurden, im nächsten Schritt ohne erneute Berechnungen als Outside deklariert werden. Entspricht die Bewegung des Benutzers nur einer Translation, dann hat sich lediglich die Entfernung aller Boxen zu den jeweiligen Ebenen der VF verändert. Diese Veränderung kann für jede Ebene durch die Projektion der Translation auf die Normale der Ebene berechnet werden und mit den Werten aus dem vorhergehenden Schritt für eine Vorverarbeitung benutzt werden. Dies funktioniert allerdings nur für ausser dem Betrachter selbst still stehende Objekte innerhalb der Szene.

Die Autoren dieser Veröffentlichung definieren dann verschiedene Szenen und verschiedene Pfade. Mit diesen werden dann jeweils die genannten Optimierungsmöglichkeiten getestet. Die Ergebnisse der Tests werden in der nächsten

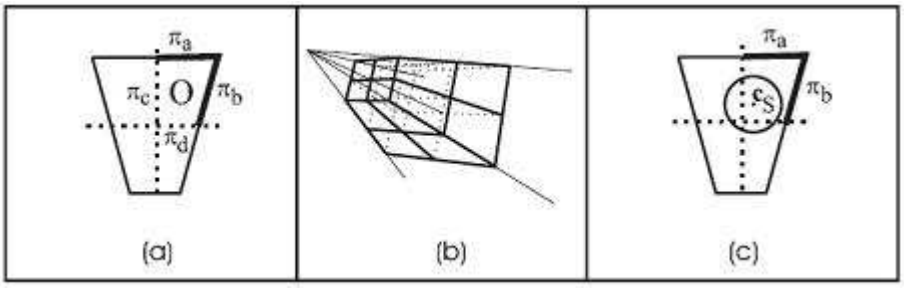


Abbildung 8.3: Unterteilung des VF in Oktanten (aus [AM00])

Tabelle dargestellt. Die enthält jeweils den Speed-Up im Vergleich zu einem Verfahren mit der eingangs schon erwähnten perspektivischen Transformation der AABB-Objekte.

Die vier in diesem Test benutzten Pfade sind wie folgt definiert:

1. Abgeleitet von den Bewegungen echter Benutzer durch die jeweilige Szene
2. Konstruiert mit jeweils einer Rotation und einer Translation in jedem Frame
3. Nur Rotationen
4. Nur Translationen

Die drei benutzten Modelle sehen wie folgt aus:

1. Modell einer Autofabrik, 184$\dot{0}$00 Polygone, 3800 Boxen
2. Modell einer Fabrik-Zelle, 167$\dot{0}$00 Polygone, 188 Boxen
3. Modell einer Fabrik, 52$\dot{0}$00 Polygone, 1274 Boxen

Anzumerken ist, dass die in diesen Tests verwendeten Boxen keine disjunkte Raumaufteilung der Szene darstellen, sondern jedes vorhandene Objekt hat eine eigene Bounding-Box.

Pfad	1	1	1	2	2	2	3	3	3	4	4	4
Modell	1	2	3	1	2	3	1	2	3	1	2	3
Nur 1	2,8	1,9	3,9	2,2	2,0	3,1	3,9	2,5	4,3	3,1	2,2	3,7
2 + 3	4,0	2,4	5,1	2,8	2,6	3,9	4,8	3,5	5,6	3,3	3,0	5,1
2 + 5	3,8	2,0	4,0	2,5	2,2	3,0	5,0	2,8	4,4	8,3	3,1	11,0
2 + 3 + 5	3,7	2,2	4,5	2,6	2,4	3,6	5,1	3,0	4,8	8,0	3,3	9,0

In der Praxis zeigte sich die Kombination aus 2 und 3, also das Abspeichern der Ebene, die für die OUTSIDE-Entscheidung gesorgt hat, und Starten des Tests im nächsten Frame mit dieser Ebene und das Einteilen des VF in Oktanten, als die beste Kombination. Allerdings nur solange man nicht nur Translationen bei den Bewegungen durch die Szene hat (Pfad 4). Das Benutzen von INSIDE-Informationen einer grösseren Box zu einer Ebene des VF bei dem Tests von den in der grösseren Box enthaltenen kleineren Boxen brachte keinen nennenswerten Vorteil und wurde daher in der Tabelle nicht erwähnt.

Wie oben schon erwähnt, bauen dieses Tests und Optimierungsmöglichkeiten darauf auf, dass zu jedem Objekt eine nicht notwendigerweise zu allen anderen Boxen disjunkte Bounding-Box definiert ist und dass über diesen Boxen noch eine Hierarchie von auch nicht notwendigerweise disjunkten grösseren Boxen aufgebaut ist. In dem Falle der Pflanzenverteilung innerhalb einer virtuellen Welt haben wir aber noch eine weitere Eigenschaft, die wir ausnutzen können. Und zwar definieren die Bounding Boxen auf der gleichen Ebene bzw. Tiefe innerhalb der Hierarchie hier eine disjunkte Raumaufteilung der ganzen Welt. Daher sollte es je nach Raumaufteilung mehr oder weniger einfach möglich sein, dass man noch Informationen zu der relativen Lage der Boxen ausnutzt.

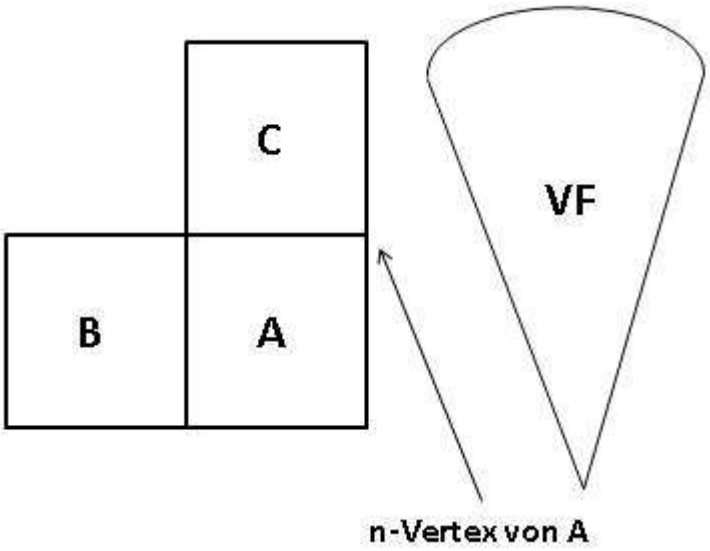


Abbildung 8.4: Möglichkeit, die Zusammenhangs-Informationen der Raumaufteilung zu nutzen

Stellen wir uns dazu mal ein Raumaufteilungsverfahren vor, bei dem jede Zelle Informationen zu ihren Nachbarzellen hat. Wurde jetzt eine Zelle aufgrund der Entfernung des n-Vertex zu einer Ebene als OUTSIDE deklariert, dann bedeutet

dies für die Nachbarzelle auf der anderen Seite, dass diese auch OUTSIDE sein wird. In dem Bild 8.4 sehen wir, dass die Zelle A aufgrund ihres n-Vertex als OUTSIDE deklariert wird, was dann für die Zelle B ebenfalls zutrifft, allerdings nicht notwendigerweise für die Zelle C. Sinnvoll wäre es dann, wenn für den Test für Zelle C dann mit der Ebene begonnen wird, die für den Ausschluss von Zelle A verantwortlich ist.

8.1.3 Pflanzenbewuchs und Ökosysteme

Ein weiterer wichtiger Punkt in virtuellen Welten und deren effizienter Umsetzung ist die Modellierung von Pflanzenbewuchs und Ökosystemen. Wir werden sehen, dass eine auf ein ganzes Ökosystem ausgerichtete Raumaufteilung viel zu der Effizienz der ganzen Welt beitragen kann. Ausserdem kann durch das Modellieren von Ökosystemen und den Einfluss, den unterschiedliche Pflanzen innerhalb des Ökosystemes aufeinander nehmen, eine noch realistischere virtuelle Welt erzeugt werden.

Ein grundlegender Aspekt bei der Entwicklung von Pflanzen im Rahmen von komplexen Ökosystemen ist die Konkurrenz dieser Pflanzen um Licht. In [Kir07] wird ein Verfahren zur Modellierung des Einflusses, den Pflanzen bei ihrer Entwicklung aufeinander haben, vorgestellt. In dem Bild 8.5 sehen wir an einem einfachen Beispiel (aus [DHL+98]), wie das Verfahren aussehen könnte. Dort wird ein Ausschnitt aus einer Szene zu verschiedenen Zeitpunkten gezeigt, jeder Punkt entspricht dem Einfluss-Bereich einer Pflanze. In dem Masse wie die Pflanzen wachsen, verdrängen sie andere Pflanzen, die dann (eventuell) absterben können. Die Farben in dem Bild bedeuten das folgende: Grün: Die Pflanze wird nicht dominiert, Rot: Die Pflanze wird dominiert, Gelb: Die Pflanze ist alt, das heisst sie hat ihre Maximalgrösse erreicht und wird in Kürze absterben.

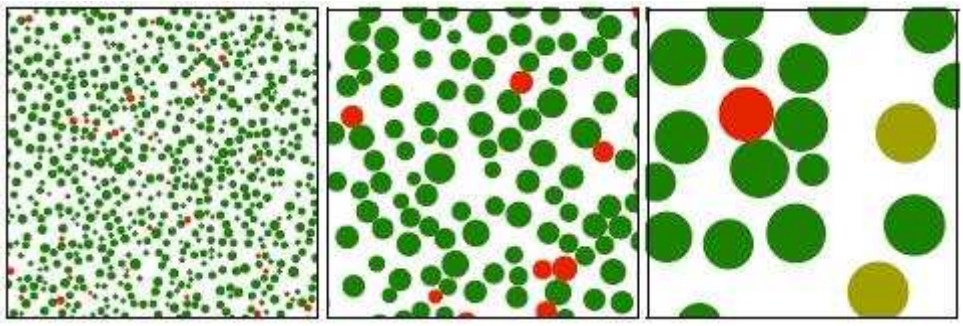


Abbildung 8.5: Entwicklung einer Pflanzen-Population (aus [DHL+98])

In [Kir07] geht es aber um die Modellierung von virtuellen Welten insgesamt. Angefangen bei der Modellierung des Terrains, kann aufbauend auf diesem Terrain sowohl die Wasserverteilung in der Welt als auch der Einfluss von Erosion berechnet werden. Mit diesen Daten kann dann ein expliziter Bezug zwischen dem Einfluss-Bereich von Pflanzen und der Vorliebe für Licht (und Schatten) sowie der Vorliebe für Wasser und besondere Bodenbeschaffenheit hergestellt werden. Der Bezug zwischen Einfluss-Bereich einer Pflanze und der Konkurrenz um Licht geht zurück auf [DHL+98], wobei Pflanzen in [Kir07] verschiedene Attribute haben und wenn sich die Einfluss-Bereiche von zwei Pflanzen überschneiden, muss dies nicht notwendigerweise zu einem Absterben einer der beiden Pflanzen führen, sondern die Pflanzen könnten trotzdem co-existieren. Im Grunde ist diese Unterscheidung sehr sinnvoll, da es ja besonders schattenliebende Pflanzen gibt und weil die gesamte Pflanzen-Vegetation sich in der Realität oft auch noch auf verschiedenen Höhenebenen befindet (Baum-Schicht, Strauch-Schicht, Kraut-Schicht und Moos-Schicht).

Es kann also mit einem recht einfachen Algorithmus dafür gesorgt werden, dass das Konkurrieren der Pflanzen um verschiedene Faktoren nachgebildet wird. Für eine sich dynamisch entwickelnde Welt kann es dann ausreichen, wenn die Parameter und Verteilung verschiedentlich aber nicht laufend geprüft wird und dann für die einzelnen Pflanzen eine entsprechende Entscheidung gefällt wird. Insgesamt reicht es aus, ein paar wenige Parameter für jede Pflanze festzulegen. Diese Parameter beschreiben im Wesentlichen die Reaktion auf Kollisionen der Einflussbereiche von Pflanzen.

Zuerst werden die Arten von möglichen Kollisionen bzw. Einflussnahmen klassifiziert, dabei werden bei [Kir07] die folgenden drei Arten unterschieden (siehe auch Bild 8.6):

Licht Die Pflanze mit dem dunkler gezeichneten Einflussbereich in Teilbild a wird nicht von der anderen Pflanze beeinträchtigt.

Halbschatten Die Pflanze mit dem dunkler gezeichneten Einflussbereich in Teilbild b wird von der anderen Pflanze beeinträchtigt, allerdings liegt der Mittelpunkt der beeinträchtigten Pflanze ausserhalb des Einflussbereiches der anderen Pflanze.

Schatten Die Pflanze mit dem dunkler gezeichneten Einflussbereich in Teilbild c wird von der anderen Pflanze sehr beeinträchtigt, da der Mittelpunkt innerhalb des Einflussbereiches der anderen Pflanze liegt

Während eines Schrittes werden nun die Pflanzen auf gegenseitige Beeinflussungen getestet, wobei ein geeignetes Raumaufteilungsverfahren sehr hilfreich ist. Zu

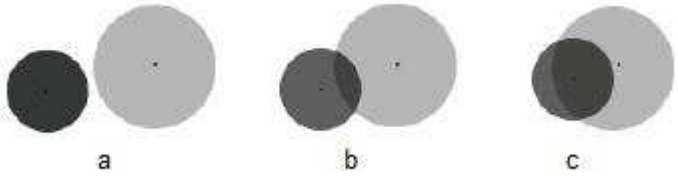


Abbildung 8.6: Möglichkeiten der Kollisionen der Einflussbereiche (aus [DHL+98])

jeder Pflanze kann nun angegeben werden, wie sie auf entsprechende Ereignisse reagiert. Die möglichen Reaktionen sehen wie folgt aus:

GROW	Wächst unbeeinflusst weiter
SLOWGROW	Das Wachstum wird verlangsamt
STOPGROW	Das Wachstum wird gestoppt
STOPMAYDIE	Das Wachstum wird gestoppt, die Möglichkeit des Absterbens besteht
DIE	Die Pflanze stirbt sicher ab
SLOWGROWMAYDIE	Das Wachstum wird verlangsamt, die Möglichkeit des Absterbens besteht
GROWMAYDIE	Das Wachstum ist normal, die Möglichkeit des Absterbens besteht

Bei Pflanzen, die den Schatten bevorzugen, könnte dann etwas wie folgt angegeben werden:

Licht	→	DIE
Halbschatten	→	SLOWGROWMAYDIE
Schatten	→	GROW

Die Vorliebe für Wasservorkommen und Bodenbeschaffenheit fliesst nicht in den oben skizzierten Algorithmus selbst ein, sondern wird bei dem Plazieren von neuen Pflanzen berücksichtigt. Anstatt Pflanzen überall in der Szene zu plazieren und zu riskieren, dass diese nach kurzer Zeit absterben, wird versucht, die Pflanzen möglichst gleich an geeigneten Positionen zu plazieren.

Dieser Ansatz wäre noch erweiterbar um die Höhen-Informationen, was dann eine zusätzliche Möglichkeit bieten würde, die Kollisions-Entscheidungen realistischer zu beeinflussen. Würde zum Beispiel in Bild 8.6 (b) der rechte grosse Einflussbereich einem 30 Meter hohen Baum gehören und der linke kleine Kreis einem Grasbüschel mit 30 Zentimeter Höhe, dann würde das Gras dem Baum wohl kein Licht wegnehmen.

In [Ham01] wird ein Ökosystem durch die Kombination aus verschiedenen Schichten modelliert. Gleichzeitig wird dort ein Zusammenhang zwischen der Grösse der Pflanzen und der Grösse der Zellen für die wünschenswerte Raumaufteilung hergestellt. In der folgenden Tabelle sehen wir die verschiedenen möglichen Schichten für eine Savannen-Landschaft.

Schicht	Zellengröße	Durchschnittliche Pflanzengröße
0	8 km	64 m
1	4 km	32 m
2	2 km	16 m
3	1 km	8 m
4	512 m	4 m
5	256 m	2 m
6	128 m	1 m
7	64 m	50 cm
8	32 m	25 cm

Zu jeder Zelle der Raumaufteilung wird danach in jeder Schicht angegeben, welche Pflanzen dort zu finden sind. Bei der Darstellung der Zelle wird zuerst berechnet, welche Schicht gerade noch zu sehen ist (basierend auf dem Blickfeld und der Entfernung) und dann nur die Pflanzen aus dieser und allen darüberliegenden Schichten angezeigt. Aufbauend auf einer solchen Verteilung unter Einbeziehung der Höhen-Informationen könnten dann bei Kollisionen Unterscheidungen gemacht werden, die dafür sorgen, dass eine Pflanze nur dann als im Halbschatten oder Schatten erkannt wird, wenn die andere an der Kollision beteiligte Pflanze auch wirklich grösser oder zumindest gleich gross ist.

Eine Aufteilung des Öko-Systemes auf die verschiedenen Schichten der Raumaufteilung hat weiterhin den Vorteil, dass für entfernte Gebiete, die zwar noch sichtbar sind, aber nicht nahe genug, um dort noch kleine Pflanzen wie Grashalme zu erkennen, nur die grossen Pflanzen gezeichnet werden müssen und für den Boden eine Texture benutzt werden könnte. Dies wird in Abbildung 8.7 illustriert, wo für die entfernten Gebiete im sichtbaren Bereich die grösseren Zellen und für die direkt vor der Kamera liegenden Gebiete die kleinsten Zellen benutzt werden. Insgesamt ist dies ein weiterer Schritt, um die Zahl der angezeigten und zu animierenden Pflanzen möglichst gering zu halten.

Trotz alledem werden insbesondere in den grossen Zellen einige grosse Pflanzen zuviel animiert und gezeichnet. Dies ist aber wahrscheinlich eher ein Vorteil, weil in den weiter entfernten Zellen werden diese grosse Pflanzen als erstes wahrgenommen. So muss bei nur kleinen Bewegungen das Blickfeld eventuell gar nicht neu berechnet werden und vielleicht auch gar keine Zellen mit Pflanzen nachgeladen werden.

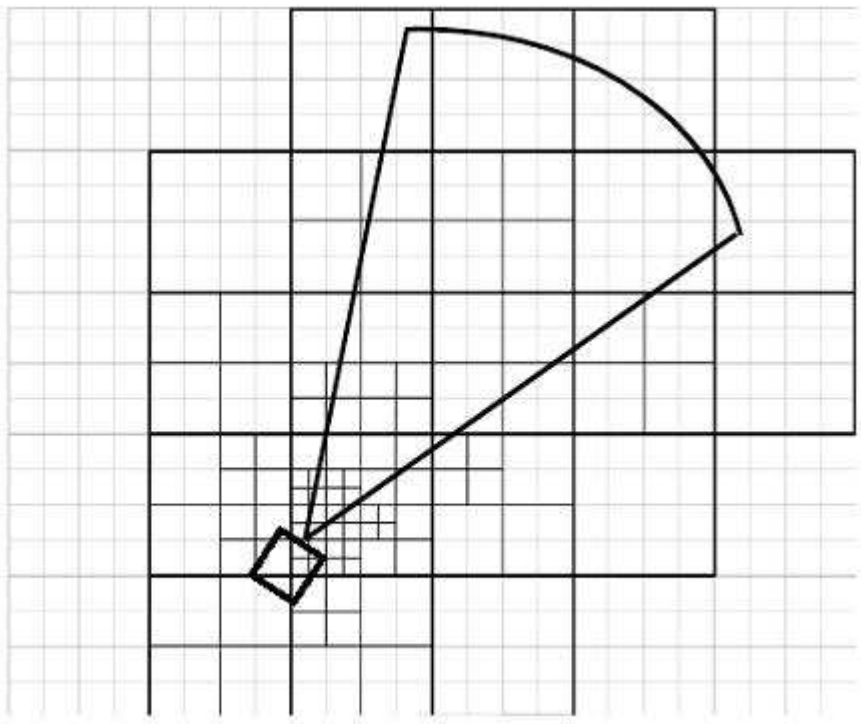


Abbildung 8.7: View Frustum in Verbindung mit den Ebenen der Raumaufteilung (aus [Ham01])

Bisher noch nicht betrachtet wurden sich schnell ändernde Parameter oder Parameter, die über einen längeren Zeitraum erfasst werden müssen. Ein sich schnell ändernder Parameter ist die Temperatur. Ein Parameter, der über einen längeren Zeitraum erfasst werden muss, ist die Anzahl der Sonnenstunden. Die Anzahl der Sonnenstunden sorgt für das Aufgehen der Blüten im Frühling und für das Abwerfen von Blättern im Herbst. Dies sind Probleme, die erst im Rahmen von einer virtuellen Welt Sinn machen, daher wollen wir hier darauf eingehen. Wir wollen uns an dieser Stelle überlegen, was möglicherweise passieren kann, zu welchen Änderungen dies führt und wie es umgesetzt werden kann. Dafür wollen wir anfangs betrachten, welche Ereignisse auftreten könnten.

Ereignis	Mögliche Umsetzung
Ein Blatt stirbt ab	Farbe oder Texture verändert sich, Parameter wie Größe oder Winkel kann sich ändern.
Ein Blatt oder Ast fällt ab	Modul entweder entfernenoder Parameter auf 0 bzw. . ganz klein setzen. Auch für alle folgenden Module. Neues und autonomes Objekt in der Szene erzeugen.
Eine Blüte öffnet oder schliesst sich	Änderung der Winkel.
Schnelleres oder langsameres Wachstum	Anpassung der Parameterentwicklung zum Beispiel durch Austausch des Funktionsalias.
Eine Blüte wird zu einer Frucht	Ersetzen des Modules, eventuell vorher eine Rückbildung von Parametern.

Es zeigt sich, dass wesentliche Aspekte der Reaktion auf die Ereignisse durch das Manipulieren der Parameter umgesetzt werden können. Verbindet man dies damit, dass die Ereignisse bei unterschiedlichen Pflanzen auch bei unterschiedlichen Werten auftreten können, legt das ganze eine Umsetzung auf der Ebene der Grammatiken nahe. Damit kann erreicht werden, dass eine Blüte sich bei einer Pflanze bei 25 Grad Celcius öffnet, während bei einer anderen Pflanze das gleiche Modul als Blüte sich erst bei 30 Grad Celcius öffnet. Der letzte Punkt, also die Tatsache, dass aus einer Blüte eine Frucht werden kann, wäre allerdings nicht nur mit Parameteränderungen realisierbar. Aber bei diesem Punkt ist auch die Erfassung des Zeitpunktes für dieses Ereignis nicht klar zu modellieren. Würde man den Zeitpunkt dieses Ereignisses nur von der Lebenszeit des Modules abhängig machen, dann bräuchte man das Modul nach seinem Lebensende nur durch ein neues Modul, welches die Frucht darstellt, zu ersetzen. Anderenfalls müsste man in der Umwelt eine Erfassung der Sonnenstunden oder Sonneneinstrahlung einführen und die Möglichkeit von Modul-Ersetzungen, die unabhängig von der Grammatik sind, in den PlantAnimator einbauen.

Für die ersten vier Ereignisse würden zusätzliche Konstrukte in der Grammatik ausreichen:

$[Reiz, Bedingung, Reaktion]$

Reiz wäre dabei ein Alias für den auslösenden Reiz (zum Beispiel TEMPERATUR), die Bedingung würde angeben, wann die Reaktion ausgeführt wird (zum Beispiel „grösser 30“) und die Reaktion würde die Möglichkeit geben, die Parameterentwicklung zu beeinflussen. Damit könnte man entweder den Texture-Parameter als Ganzes auszutauschen, oder die Entwicklung von anderen Parameter zu steuern. Um Entwicklung eines Parameters zu ändern, müsste man ein neues Element $[x, y, z, b, e]$ angeben. Dabei sollte sich der neue Startwert x an

dem zu diesem Zeitpunkt aktuellen Wert des auszutauschenden Parameters orientieren.

8.1.4 Movement Prediction

Nachdem nun klar ist, was eine virtuelle Welt ist und wie das Blickfeld modelliert und berechnet werden kann, sollte noch geklärt werden, wie und ob eine Vorhersage der Bewegungen des Betrachters effizient erfolgen kann. Diese Vorhersage würde benötigt werden, um dann die sogenannte Lazy Animation noch besser in Angriff nehmen zu können. Eine geeignete Vorhersage der Bewegungen erlaubt das frühzeitige Anstossen der Aktualisierung von bisher inaktiven Pflanzen und vermeidet damit wahrgenommene Artefakte bei Drehungen des Kopfes oder Laufen durch die Welt. Es muss also betrachtet werden, wie man die Bewegungen des Benutzers (auch unter Zuhilfenahme der bisherigen Bewegungen) möglichst einfach vorhersagen kann.

Existierende Ansätze in diesem Gebiet sind meistens der künstlichen Intelligenz zuzurechnen. Es werden zum Beispiel Ansätze über Neuronale Netze (siehe [GV06]) oder Hidden Markov Modelle (siehe [VGPU04]) benutzt, um aufbauend auf Trainingsdaten ein Verfahren anzulernen, dass die nächsten Bewegungen vorhersagen kann.

Obwohl diese komplexe Vorverarbeitung und das Trainieren des Verfahrens zuerst abschreckt ist der eigentlich grosse Nachteil dieser Verfahren, dass sie stark von den Trainingsdaten und damit von der zugrundeliegenden Welt oder Szene abhängen. Da es wahrscheinlich nur schwer möglich sein wird, möglichst allgemeingültige Trainingsdaten zu bekommen, wird es wohl nötig sein, eine möglichst einfache Vorhersage über die folgenden Bewegungen auf Basis der konkreten bisherigen Bewegung zu treffen. Dabei sollte es ein Ziel sein, dass bestimmte Bewegungsabläufe von vorneherein als unnatürlich oder zumindest für Menschen schwer möglich auszuführen erkannt werden.

Die Herausforderung bei der Movement Prediction sind schnelle Kopfdrehungen, betrachten wir dazu beispielsweise noch einmal das Bild 8.7, die kleinste Zelle hat eine Grösse von 32 Metern und nach spätestens 150 Metern kann ein Mensch schon die Pflanzen dieser Schicht nicht mehr erkennen. Würde der Mensch an der Kameraposition sich nun nach vorne bewegen, dann würde er für die 32 Meter selbst im schnellsten Hundertmeter-Sprint um die 3 Sekunden und damit etwa 72 Bild-Aktualisierungen benötigen. Esrt dann kommt er in der nächsten Zelle an und es müssen neue Zellen in Sichtweite geladen werden. Dreht er allerdings seinen Kopf recht schnell, so werden in wesentlich kürzerer Zeit mehr neue Zellen in sein Blickfeld gelangen.

Und genau das ist ein echtes Problem. Bei Bewegungen eines Menschen durch Gehen oder Laufen lassen sich je nach Geschwindigkeit gewisse Bewegungsmuster sicher ausschliessen. Als Beispiel sei hier eine 90 Grad Drehung der Laufrichtung in vollem Hundermeter-Sprint genannt. Kopfbewegungen und Rotationen sind allerdings nur schwer vorhersagbar. Hier kann man bestenfalls auf einfache Weise eine Art Vorhersage treffen. Hat sich der Kopf in den letzten Frames immer mit einer gewissen Geschwindigkeit in eine bestimmte Richtung (links oder rechts) gedreht, dann wird es in den nächsten Frames wohl in die gleiche Richtung weitergehen. Allerdings nur solange gewisse Gültigkeits-Kriterien erfüllt bleiben, da ein Mensch seinen Kopf schliesslich nur bis zu einer gewissen Grenze drehen kann.

8.2 Level of Detail für die Animation

Eine weitere sehr bekannte Optimierungsmöglichkeit in grossen Szenen ist eine Level of Detail Darstellung. Üblicherweise versteht man unter Level of Detail (LoD), dass ein Objekt in der Szene durch verschieden komplexe geometrische Darstellung repräsentiert wird. Welche Darstellung dann tatsächlich aktuell in der Szene benutzt wird, wird normalerweise anhand der Entfernung zwischen Betrachter und Objekt ausgewählt.

Zusätzlich dazu oder auch in Verbindung damit, bietet das Konzept des PlantAnimator auch eine Level of Detail Aufteilung für die Animation. In Abhängigkeit der Entfernung zwischen Betrachter und Pflanze kann unterschiedlich viel Aufwand in die Animation der Pflanze gesteckt werden. Eine Pflanze im PlantAnimator wird aus einer Hierarchie von Modulen zusammengesetzt und es gibt einen Wert bis zu welcher Tiefe die umwelt-getriebene Animation aufbauend auf der aktuellen Richtung und Stärke des Windes wirklich berechnet wird und demzufolge auch ab welcher Tiefe nur noch eine zufällige Bewegung der Module auftritt.

Das Konzept kann noch etwas verfeinert werden, indem man eine zweite Grenze über dieser ersten Grenze einführt. Ab dieser zweiten Grenze wird vielleicht gar nicht mehr animiert und Bewegungen für diese Module ergeben sich nur noch durch Bewegungen der animierten und in der Hierarchie vorher kommenden Module.

Durch die Angabe dieser Grenzen und auch die Möglichkeit, diese Grenzen für jede Pflanze auf Basis der aktuellen Entfernung zwischen der Pflanze und dem Betrachter dynamisch von Frame zu Frame zu verändern ohne wissen zu müssen, wie das Konzept innerhalb des PlantAnimators umgesetzt wird, kann nun ähnlich dem recht bekannten Level of Detail für diestatische Darstellung von Objekten, eine Level of Detail Darstellung für die Animation definiert werden. Diese kann

Hand in Hand gehen mit der LoD für die statische Darstellung (naturgemäss wird eine Pflanze, deren geometrische Darstellung aufgrund der normalen LoD-Darstellung einfacher geworden ist, auch weniger Rechenzeit für das Berechnen der umwelt-getriebenen Animation benötigen). Sie kann aber auch völlig losgelöst von dieser betrachtet werden, d. h. die Genauigkeit der Animation kann geändert werden, ohne die statische Darstellung eines Objektes zu verändern.

Dies könnte auch verbunden werden mit der hierarchischen Raumaufteilung der gesamten Pflanzenwelt in der Szene. Wir erinnern uns, dass unterschiedlich grosse Pflanzen den unterschiedlichen Ebenen der Raumaufteilungs-Hierarchie zugeordnet werden. Wird ein hoher Baum der sechsten Schicht in einer Zelle dargestellt, von der nur die dritte Schicht sichtbar ist (dort, wo die Zelle eine Seitenlänge von 1 Kilometer hat und der Baum wohl sicher auch entsprechend weit entfernt ist), dann reicht vielleicht etwas zufälliges Bewegen des Stammes und der ersten wenigen Äste aus, um den Baum als animiert erscheinen zu lassen. Befindet sich der gleiche Baum näher am Betrachter, das heisst er wird dargestellt, neben ihn herum werden aber auch Pflanzen der achten Schicht dargestellt, wo die Seitenlänge einer Zelle nur noch 32 Meter ist und der Baum sich sicher dementsprechend nahe an dem Benutzer befindet, muss wahrscheinlich mehr Aufwand in die Animation des Baumes gesteckt werden.

Eine solche Koppelung zwischen Raumaufteilung für die Darstellung der sichtbaren Pflanzen und der Bestimmung der Grenzen für die Animation scheint eine gute Lösung zu sein. Noch vielversprechender scheint eine Koppelung zwischen hierarchischer Raumaufteilung, einer Level of Detail Darstellung für die Pflanzen-Topologie (also die Komplexität der benutzen Grammatik) und dem Level of Detail Konzept der Animation. Da sich allerdings aus den Definitionen des PlantAnimators schon Möglichkeiten zur Einsparung der für eine Pflanze zu speichernden Informationen ergibt (zum Beispiel mit SharedGeometry und durch die Quantisierung), kann eventuell auf die Level of Detail Darstellung mit verschieden komplexen Grammatiken verzichtet werden. Der Sinn der Level of Detail Darstellung mit verschieden komplexen Grammatiken besteht in dem Einsparen von benötigtem Speicherplatz. Eine solche Darstellung würde es aber auch nötig machen, dass bei den Übergängen jeweils eine alte Pflanzendarstellung aus der Szene entfernt und eine neue hinzugefügt werden muss. Im Gegensatz dazu muss bei dem Level of Detail für die Animation lediglich dynamisch die Grenzen für die Animationstiefe geändert werden.

8.3 Lazy Animation

Nachdem wir bisher in diesem Kapitel gesehen haben, was alles gemacht werden kann, um bei allen Pflanzen in der Szene eine möglichst effiziente Unterteilung in sichtbare und nicht sichtbare Pflanzen zu erreichen und die sichtbaren Pflanzen abhängig von dem Grad der Sichtbarkeit auch noch mit unterschiedlich viel Aufwand animieren zu können, geht es in diesem letzten Abschnitt um die Frage und Behandlung der nicht sichtbaren Pflanzen. Wünschenswert ist hier eine Lösung, die es erlaubt möglichst wenig oder vielleicht sogar gar keinen Rechen-Aufwand in nicht sichtbare Pflanzen zu stecken. Gleichzeitig muss es aber auch möglich sein, dass Pflanzen, die wieder in den Sichtbarkeits-Bereich kommen, schnell aktualisiert werden können.

Der PlantAnimator und die Grammatiken sind nun so gebaut, dass sie diese Lazy Animation erlauben. Dabei geht es nicht darum, dass eine Pflanze kontinuierlich und fortwährend animiert wird (um zum Beispiel auch zu wachsen), sondern es gibt eine äussere Instanz, welche Wissen darüber hat, ob eine Pflanze gerade sichtbar ist bzw. animiert werden muss. Diese Instanz sendet regelmässige Signale an alle sichtbaren Pflanzen, damit diese sich Aktualisieren und Zeichnen können. Ist eine Pflanze nicht sichtbar, so bekommt sie keine Signale gesendet. Kommt die gleiche Pflanze aber durch einen Kameraschwenk oder ähnliches in den Sichtbarkeitsbereich hinein, so muss diese Pflanze schnell aktualisiert werden. Dies muss möglich sein, ohne die ganze Arbeit, die zwischendurch eingespart wurde, auszuführen. Das System des PlantAnimator eignet sich dafür gut, da dort die Zeitachse besonders betont ist. Daher kann auf einfache Art bestimmt werden, welche Ersetzungen im Sinne der Grammatik vorgenommen werden müssen. Danach werden die für die Animation nötigen Parameter (wie Länge und Breite) einmal neu bestimmt. Im Gegensatz zu beispielsweise den differentiellen L-Systemen, wo eventuell erst noch die Differentialfunktionen mehrfach ausgewertet werden müssen, bevor eine Regel (in Abhängigkeit von dem Parameter, der dann aus seiner Domain herausgelaufen ist) angewendet werden kann. Dadurch spart diese Vorgehensweise fast die gesamte Arbeit, die angefallen wäre, während die Pflanze nicht sichtbar gewesen ist.

Ein Problem bleibt allerdings das Aktualisieren einiger Parameter, sobald eine Pflanze wieder in den Sichtbarkeits-Bereich kommt. Das Auswerten der Parameter für Länge, Breite und Winkel ist dabei kein Problem und erzeugt genauso viel Aufwand, wie ein normaler Animationsschritt erzeugt hätte. Kritisch sind vielmehr der Aufwand, der sich durch während der Phase der Inaktivität hinzugekommene Module ergibt, und der Anfangszustand für die umwelt-getriebene Animation.

Der Aufwand durch neu hinzugekommene Module wird dabei moderater sein, als

zuerst vermutet, da der PlantAnimator mit Modul-Prototypen und SharedGeometry arbeitet. Neu hinzugekommene Module benötigen also im Normalfall nicht das Anlegen einer ganz neuen und komplexen Geometrie. Für die Bestimmung des Anfangszustandes für die umwelt-getriebene Animation (also der beiden Parameter a und b, wie in Abschnitt 6.4.2 beschrieben) sind verschiedene Varianten denkbar, unter Umständen auch wieder darauf basierend, in welcher Entfernung die Pflanze in den Sichtbarkeits-Bereich hereingekommen ist.

- Man berechnet a und b für die aktuelle Windrichtung und startet die umwelt-getriebene Animation mit $\frac{a}{2}$ und $\frac{b}{2}$. Dies ist sicher für weiter entfernte Pflanzen eine effiziente Möglichkeit, die umwelt-getriebene Animation neu zu starten.
- Man berechnet a und b für die aktuelle Windrichtung und a_{alt} und b_{alt} für die Windrichtung vor einer bestimmten Anzahl x von Frames. Die umwelt-getriebene Animation wird dann mit den Werten $a_{alt} + \frac{a - a_{alt}}{2}$ und $b_{alt} + \frac{b - b_{alt}}{2}$ gestartet.
- Das Verfahren aus Punkt 2 könnte eventuell noch auf mehrere Zwischenschritte erweitert werden, also nicht nur für das x. Frame vor dem aktuellen, sondern auch noch für das (2*x). und (3*x). Frame durchgeführt werden.

Mit solchen Berechungen müsste es möglich sein, neue Anfangswerte für die umwelt-getriebene Animation effizient zu bestimmen.

Um noch einmal auf das Anlegen von neuen Modulen aufgrund von Pflanzenwachstumes zurückzukommen, dieses kann eventuell noch beschleunigt werden, indem der Scheduler Phasen nutzt, in denen der Benutzer sich wenig bis gar nicht bewegt. Durch die Betonung der Zeitachse müsste es nämlich gut möglich sein, dass der Scheduler auch für inaktive Pflanzen eine Vorhersage treffen kann, wann und wie diese neue Module durch Wachstum bekommen. So könnte das Anlegen von neuen Modulen schon im Hintergrund geschehen in Phasen mit wenig Bewegungen des Benutzers.

Kapitel 9

Zusammenfassung und Ausblick

9.1 Zusammenfassung

In dieser Arbeit geht es um die Modellierung und Animation von computergenerierten Pflanzen. In einem ersten Schritt wurde ein Verfahren (im folgenden PlantAnimator genannt) vorgestellt. Mit dem PlantAnimator ist es möglich, sowohl computergenerierte Pflanzen auf einfache, intuitive und auch sehr vielfältige Weise zu modellieren, als auch Wachstumsaspekte der Animation mitzumodellieren. Um mit dem PlantAnimator eine Pflanze zu modellieren, muss man zuerst die verschiedenen Module konstruieren. Ein Modul besteht hierbei aus der Geometrie und dem Aussehen. Die definierten Module werden dann anhand einer einem L-System ähnlichen Grammatik zu einer Gesamtpflanze zusammengesetzt. Dabei spielen Animations-Aspekte und Stochastik eine grosse Rolle.

Schon bei diesem ersten Ansatz wurde sehr stark auf die Effizienz des Verfahrens geachtet und auf Möglichkeiten der Optimierungen wie die Quantisierung oder die Umsetzung eines SharedGeometry-Konzeptes viel Wert gelegt, da dies im Rahmen der computergenerierten Pflanzen und insbesondere, wenn man an virtuelle Welten mit tausenden Pflanzen denkt, ein besonders wichtiger Aspekt ist. Meist ist die Effizienz sogar noch wichtiger als die botanisch oder physikalisch völlig korrekte Umsetzung der Realität.

Aufbauend auf diesem ersten Schritt wurde dann noch weitere Schritte unternommen, um die Konzepte in Richtung einer umfassenden und effizienten Umsetzung der Animation zu erweitern. Das Ziel sollte eine Untersuchung und die Vorstellung von Ideen sein, wie die Animation von computergenerierten Pflanzen in virtuellen Welten aussehen und verbessert werden kann, was wohl den Schwerpunkt des Einsatzes von computergenerierten Pflanzen ausmachen wird.

Zuallererst wurde daher betrachtet, wie man die umwelt-getriebene Animation in das bestehende Konzept des PlantAnimators einfügen kann und es wurden weiterhin noch Verbesserungen des ersten und einfachen Systemes vorgeschlagen. Diese Verbesserungen reichen von dem Einführen einer einfachen Rekursion bei dem Modellieren bis hin zu kleinen Schönheitsreparaturen, wie dem automatischen Berechnen von kleinen Zwischenobjekten an den Übergängen zwischen Modulen.

Gegen Ende der Arbeit wurden verschiedene Konzepte, die im Rahmen von virtuellen Welten besonders wichtig sind, vorgestellt. Es wurde versucht darzulegen, wie diese Konzept ineinandergreifend und effizient umgesetzt werden können, bevor es noch einmal daran ging aufzuzeigen, welche weiteren Optimierungsmöglichkeiten der PlantAnimator bei dem Einsatz in virtuellen Welten bietet (namentlich Level of Detail für die Animation und Lazy Animation). So sorgt unter anderem das starke Betonen der Zeitachse bei dem Modellieren im PlantAnimator (im Gegensatz zu beispielsweise den differentiellen L-Systemen), dass es möglich ist, eine Pflanze eine Weile lang gar nicht zu aktualisieren und zu animieren, dies aber sehr effizient nachholen zu können, sobald diese wieder in das Blickfeld kommt.

Es wurde zwar deutlich, dass die Strategie des PlantAnimators nicht an allen Stellen die beste Lösung ist, aber in Punkto Effizienz des Verfahrens wohl kaum verbessert werden kann. Insbesondere bei der Umsetzung der umwelt-getriebenen Animation scheint zumindest ein vertex-basiertes Verfahren besser für ein noch etwas realistischeres Aussehen geeignet zu sein, allerdings ist beispielsweise die Idee, die Windrichtung auf die Grundebene des Modules zu projezieren, um ein Mass für Stärke und Richtung der Wind-Einwirkung zu haben, auch für ein vertex-basiertes Verfahren sicher nicht verkehrt.

Letztendlich wurden viele für die Modellierung Animation von computergenerierten Pflanzen wichtigen Aspekte betrachtet und Lösungen für verschiedene Aspekte durch eigene Ideen oder die Kombination von vorhandenen Verfahren unter Zusatz von eigenen Ideen vorgeschlagen.

9.2 Ausblick

In diesem letzten Abschnitt geht es darum, verschiedene weitere Verbesserungsmöglichkeiten aufzuzeigen oder noch einmal Punkte herauszustellen, deren geeignete Bearbeitung dann doch den Rahmen dieser Arbeit gesprengt hätten, die aber für die Zukunft vielversprechend erscheinen.

- Ähnlich wie die Unterscheidung in Forward und Inverse Kinematic, gibt es auch eine Art inverse umwelt-getriebene Animation. Diese nennt sich für die Animation allgemein Inverse kinematic animation und das Ziel ist, dass eine Position für das Ende einer Kette aus verschiedenen Gliedern und Gelenken vorgegeben wird und daraus dann gültige Positionen für alle Glieder und Gelenke der Kette zurückgerechnet werden können. Ein solches Verfahren in die Ideen des PlantAnimators zu integrieren, scheint sinnvoll, wenn es beispielsweise darum geht, einen Menschen, der durch hohes Gras läuft modellieren zu wollen. Der Mensch wird die Grashalme beseite drücken und im schlimmsten Falle abbrechen. Ausserdem könnte die Möglichkeit, Positionsänderungen auch top-down durch die Pflanze vorzunehmen, auch helfen, wenn es darum geht, mit Pflanzen umzugehen, deren Stämme und Stengel wesentlich dünner als die Blätter und Blüten sind, also wo der Einfluss des Windes wesentlich von den Blüten und Blättern abhängt. Das Entwickeln einer entsprechenden Erweiterung des PlantAnimators wäre daher sehr wünschenswert.

- Die Möglichkeiten der Unterstützung der Verfahren durch die GPU sollte untersucht werden. Dabei sollten wir im Hinterkopf haben, dass es an der einen oder anderen Stelle sinnvoll sein kann, auf der Basis von Vertices zu arbeiten. Die Frage ist, ob man dies effizient in den PlantAnimator integrieren kann, ohne Vorteile wie die mögliche Quantisierung und das Nutzen von SharedGeometry zu verlieren.

- Die Effizienz des PlantAnimator hängt wesentlich von dem Benutzen von affinen Transformationen ab. Insgesamt werden viele affine Transformationen bei der Visualisierung benutzt. Ein Modul innerhalb einer Pflanze bekommt auf jedenfall die affine Transformation seines Vorgänger-Modules, dazu aber sicher noch mindestens zwei Skalierungen, eine Rotation, eine Translation und eine Scherung. Eine Untersuchung der Möglichkeiten, diese oder allgemein mehrere affine Transformationen möglichst effizient zusammenrechnen zu können, scheint daher für den PlantAnimator sinnvoll zu sein.

- Es könnte weiterhin interessant sein, zu untersuchen inwieweit sich ein Konzept zur Kollisionserkennung in den PlantAnimator integriert werden kann. Aus Effizienz-Gründen und weil es für eine erste realistische Darstellung

nicht zwangsläufig nötig ist, wird in dem PlantAnimator keine Kollisionserkennung integriert. Trotzdem wäre es wahrscheinlich sinnvoll, wenn man untersucht, wie die Interaktion mit einer Kollisionserkennung für bewegliche Objekte aussehen kann (eventuell nur für die ersten Ebenen der Modul-Hierarchie ähnlich der Umsetzung der umwelt-getriebenen Animation).

- Wir haben gesehen, wie das Berechnen des Blickfeldes aussehen kann. Dies passiert typischerweise auf Basis der in der Szene enthaltenen Objekte ohne dabei notwendigerweise eine vollständige und disjunkte Raumaufteilung der gesamten Szene zu erhalten. Wie wir aber in Bild 8.4 auf Seite 144 gesehen haben, könnte es sein, dass man die Informationen, die implizit in einer vollständigen und auf den einzelnen Ebenen disjunkten Raumaufteilung stecken, auch zur noch effizienteren Berechnung des Blickfeldes führen.

- Die Auswirkungen von Parametern in der Umwelt, die sich entweder recht schnell ändern können oder noch schlimmer über einen grösseren Zeitraum betrachtet werden müssen (wie zum Beispiel die Anzahl der Sonnenstunden), wurde nicht tiefgehend untersucht und daher auch nicht vollständig in den PlantAnimator integriert.

- In einer virtuellen Welt spielt Movement Prediction eine durchaus wichtige Rolle. Auf Basis der Vorhersage kann möglichst frühzeitig eine Entscheidung getroffen werden, welche Teile der Welt nachgeladen oder aktiviert werden müssen, damit der Benutzer möglichst nicht merkt, dass nicht die ganze Welt gleichzeitig animiert wird. Movement Prediction wird bisher vorzugsweise im Rahmen der künstlichen Intelligenz eingesetzt. Dort gibt es Verfahren, die beispielsweise auf neuronalen Netzen oder Hidden Markov Modellen basieren, um aufbauend auf Trainingsdaten zu einer bestimmten Szene eine Bewegungs-Vorhersage für diese Szene treffen zu können. Da es für allgemeine virtuelle Welten sicher nur schwer möglich ist, allgemeingültige Trainingsdaten zu bekommen, eignen sich solche Konzept nur für einzelne virtuelle Welten von denen Trainingsdaten existieren. Die Frage wäre, ob man nicht auch zumindest ein einfaches Verfahren entwerfen kann, welches auf den bisher stattgefundenen konkreten Bewegungen des aktuellen Benutzers beruht und dann versucht aufgrund von Vorüberlegungen gewisse Bewegungsmuster (wie eine sehr grosse Richtungsänderung von etwa 90 Grad bei schnellem Rennen) auszuschliessen oder besonders wahrscheinliche Bewegunsgmuster herauszufinden. Die Frage, ob dabei das Terrain einen Ausschlag für mögliche Bewegungsrichtungen geben kann, wurde beispielsweise schon in [Bis01] angedacht.

- In dieser Arbeit wurde eine Formel für die Stärke s des Wind-Einflusses auf die Pflanzen vorgeschlagen, die auf dem Durchmesser und der Festigkeit des Pflanzen-Materiales beruht. Das Überprüfen oder praktische Belegen

der Korrektheit der Formel war allerdings im Rahmen dieser Arbeit nicht möglich, wäre aber grundsätzlich schon von Interesse. So sind wir hier davon ausgegangen, dass ein grösserer Durchmesser von Anfang an die Windbeeinflussbarkeit senkt, wollen uns dem Gedanken, dass diese Abnahme erst ab einem gewissen Minimalwert wirklich greift und bis dahin die vergrösserte Angriffsfläche für den Wind mehr Einfluss hat, nicht grundsätzlich verschliessen.

- Es könnte interessant sein, zu untersuchen, welche der notwendigen Berechnungen eventuell auf andere Rechner ausgelagert werden können. Dabei wird es von Interesse sein, zu untersuchen, in welchem Kontext beispielsweise eine virtuelle Welt eingesetzt wird. Je nachdem, ob der Client in einer Client-Server-Architektur als thin oder fat eingesetzt wird, kann diese Idee von Interesse sein. In [Kle03] wurde gezeigt, dass im Rahmen eines fat Clientes die Sichtbarkeits-Berechnungen jeweils auf den Client der einzelnen Mitspieler in einem Online-Rollenspiel ausgelagert werden können. Im Rahmen der computergenerierten Pflanzen könnte vielleicht ein thin Client besonders interessant sein, wenn es möglich ist die für die Bewegung der Pflanzen notwendigen Berechnungen so auszugliedern, dass diese nur einmal auf dem Server (oder auf einem zugeordneten Sub-Server) ablaufen können. Und am besten ist vielleicht die Kombination aus beiden Verfahren, d. h. die Sichtbarkeits-Berechnungen könnten auf dem Client durchgeführt werden, welcher sich dann die aktualisierten Daten der anzuzeigenden Pflanzen von dem Server holt. Da sich die Geometrie-Daten der Module und Pflanzen im Rahmen des PlantAnimators nicht ändern, könnten die beiden Verfahren vielleicht mit wenig Daten-Transfer zwischen Client und Server kombiniert werden.

- Neben dem Wind gibt es noch andere Arten von umwelt-getriebener Animation. Dies könnte zum Beispiel eine Strömung in einem Meer oder Fluss sein, die darin befindliche Pflanzen bewegt. Der Unterschied zwischen Wind und Strömung wäre im Wesentlichen, dass der Einfluss der Strömung wesentlich nachhaltiger ist als der des Windes, von daher wird es an der Stelle wahrscheinlich nicht ratsam sein, mit einem grösserem zufälligem Anteil in der Bewegung durch Wasserströmungen zu arbeiten. Das Zusammenstellen und der Vergleich der möglichen Arten von umwelt-getriebener Animation und das Untersuchen der möglicherweise anderen Anforderungen, sowie das Betrachten wie diese Konzepte allgemein noch berücksichtigt werden können, wäre eine interessante Fragestellung.

Literaturverzeichnis

[AM00] Ulf Assarsson und Tomas Möller: *Optimized view frustum culling algorithms for bounding boxes.* Journal of Graphics Tools, 5:9–22, 2000.

[And06] Olga Andriyenko: *Interaktive Algorithmen zur Modellierung von Pflanzenwachstum*, 2006.

[Bis01] I. D. Bishop: *Predicting movement choices in virtual environments.* Landscape and Urban Planning, Seiten 97–106, 2001.

[CS06] Eric M. Church und SK Senwal: *Simulating Trees using Fractals and L-Systems.* In: *Proceeding (556) Environmental Modelling and Simulation*, 2006.

[Deu03] Oliver Deussen: *Computergenerierte Pflanzen.* Springer, 2003.

[DHL+98] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Měch, Matt Pharr und Przemyslaw Prusinkiewicz: *Realistic modeling and rendering of plant ecosystems.* In: *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, Seiten 275–286, New York, NY, USA, 1998. ACM.

[DL97] Oliver Deussen und Bernd Lintermann: *Erzeugung komplexer botanischer Objekte in der Computergraphik.* Informatik Spektrum, 20(4):208–215, 1997.

[dREF+88] Phillippe de Reffye, Claude Edelin, Jean Françon, Marc Jaeger und Claude Puech: *Plant models faithful to botanical structure and development.* In: *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, Seiten 151–158, New York, NY, USA, 1988. ACM.

[GV06] Arpad Gellert und Lucian Vintan: *Person Movement Prediction Using Hidden Markov Models*, 2006.

[Ham01] Johan Hammes: *Modeling of Ecosystems as a Data Source for Real-Time Terrain Rendering.* In: *DEM '01: Proceedings of the First International Symposium on Digital Earth Moving*, Seiten 98–111, London, UK, 2001. Springer-Verlag.

[Kir07] Norman Kirch: *Modellierung von virtuellen Welten*, 2007.

[Kle03] Marcus Klein: *Optimierung des Informationsaustausches bei Massive Multi-Player Online Games*, 2003.

[KP03] Radoslaw Karwowskia und Przemyslaw Prusinkiewicz: *Design and Implementation of the L+C Modeling Language.* 4th International Workshop on Rule-Based Programming, 2003.

[LD99] Bernd Lintermann und Oliver Deussen: *Interactive Modeling of Plants.* IEEE Comput. Graph. Appl., 19(1):56–65, 1999.

[May06] Maya: *MAYA Tutorial - Basics of Animation*, 2006.

[MP96] Radomír Měch und Przemyslaw Prusinkiewicz: *Visual models of plants interacting with their environment.* In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, Seiten 397–410, New York, NY, USA, 1996. ACM.

[NRW08] Learn:Line NRW: *http://www.learnline.nrw.de/angebote/modell/runge.htm*, 2008.

[NTT92] Hansrudi Noser, Daniel Thalmann und Russell Turner: *Animation based on the Interaction of L-systems with Vector Force Fields.* In: *Proc. Computer Graphics International '92*, Seiten 747–761. Springer, 1992.

[Opp86] Peter E. Oppenheimer: *Real time design and animation of fractal plants and trees.* SIGGRAPH Comput. Graph., 20(4):55–64, 1986.

[Par02] Rick Parent: *Computer animation: algorithms and techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[PHM93] Przemyslaw Prusinkiewicz, Mark S. Hammel und Eric Mjolsness: *Animation of plant development.* In: *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, Seiten 351–360, New York, NY, USA, 1993. ACM.

[PHM00] Przemyslaw Prusinkiewicz, Jim Hanan und Radomír Mech: *An L-System-Based Plant Modeling Language.* In: *AGTIVE '99: Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance*, Seiten 395–410, London, UK, 2000. Springer-Verlag.

[PJM94] Przemyslaw Prusinkiewicz, Mark James und Radomír Měch: *Synthetic topiary.* In: *Computer Graphics Proceedings, Annual Conference Series, 1994*, Seiten 351–358. ACM Press, 1994.

[PL96] Przemyslaw Prusinkiewicz und Aristid Lindenmayer: *The algorithmic beauty of plants.* Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[Pur07] Werner Purgathofer: *http://www.cg.tuwien.ac.at/courses/CG/textblaetter/04 Geometrische Transformationen.pdf*, 2007.

[RB85] William T. Reeves und Ricki Blau: *Approximate and probabilistic algorithms for shading and rendering structured particle systems.* SIGGRAPH Comput. Graph., 19(3):313–322, 1985.

[Ree83] William T. Reeves: *Particle Systems - A Technique for Modeling a Class of Fuzzy Objects.* ACM Transactions on Graphics, 2:359–376, 1983.

[Sch97] Hans Rudolf Schwarz: *Numerische Mathematik.* Teubner, Stuttgart, 4. Auflage, 1997.

[Sta97] Jos Stam: *Stochastic dynamics: Simulating the effects of turbulence on flexible structures.* Computer Graphics Forum, 16:159–164, 1997.

[VGPU04] Lucian Vintan, Arpad Gellert, Jan Petzold und Theo Ungerer: *Person Movement Prediction Using Neural Network*, 2004.

[Wik07] Wikipedia: *http://en.wikipedia.org/wiki/Viewing_frustum*, 2007.

[Wik08a] Wikipedia: *http://de.wikipedia.org/wiki/Brownsche_Bewegung*, 2008.

[Wik08b] Wikipedia: *http://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens*, 2008.

[Wik08c] Wikipedia: *http://de.wikipedia.org/wiki/Pflanzenbewegung*, 2008.

[Wik08d] Wikipedia: *http://de.wikipedia.org/wiki/Sigmoidfunktion*, 2008.

Ich erkläre, daß ich die vorliegende Arbeit selbstständig verfaßt habe und keine anderen als die angegebenen Hilfsmittel verwendet habe. Für Textstellen, die wörtlich oder sinngemäß aus anderer Literatur übernommen worden sind, werden stets die Quellen angegeben. Ausserdem erkläre ich, dass ich die Dissertation in der vorliegenden oder einer ähnlichen Form bisher noch nicht zu Prüfungszwecken eingereicht habe.

Marburg, im Oktober 2008